НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено

Завідувач кафедри

_____ О.В. Коваль
(підпис) (ініціали, прізвище)

“___”_____2019р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки 6.050101 “ Комп’ютерні науки ”

на тему Деформація растрових зображень для нанесення на поверхні

Виконав: студент 4 курсу, групи ТР-51

Заковоротний Олександр Ігорович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник д.т.н. Аушева Наталія Миколаївна _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 6.050101 “ Комп’ютерні науки”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2019р.

ЗАВДАННЯ
на дипломну роботу студенту
Заковоротному Олександру Ігоровичу
(прізвище, ім’я, по батькові)

1. Тема роботи “Деформація растрових зображень для нанесення на поверхні”

керівник роботи д.т.н. Аушева Наталія Миколаївна
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від 22.05.2019р. №1325-С

2. Строк подання студентом роботи _____
3. Вихідні дані до роботи послідовність деформованих зображень для нанесення на поверхню анімації морфінгу
4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати існуюче програмне забезпечення та методи для деформації зображення, обґрунтувати обрані засоби та методи для програмної реалізації, створити алгоритмічну базу для реалізації деформації зображення вибраними засобами та методами, розробити програмний продукт для створення анімації морфінгу та перенесення її на поверхню.
5. Перелік ілюстративного матеріалу
1. Визначення деформації зображення. 2. Методи та засоби для деформації зображення. 3. Анімація морфінгу двох зображень. 4. Методи задання поверхонь. 5. Деформація зображення для нанесення на поверхні. 5. Архітектура системи. 6. Інтерфейс користувача. 7. Результати дослідження методів деформації.
6. Дата видачі завдання ”___”_____201___ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи		
2.	Вивчення та аналіз задачі		
3.	Розробка архітектури та загальної структури системи		
4.	Підготовка матеріалів		
5.	Програмна реалізація системи		
6.	Оформлення пояснювальної записки		
7.	Захист програмного продукту		
8.	Передзахист		
9.	Захист		

Студент _____ Заковоротний О. І.
(підпис) (прізвище та ініціали,)

Керівник роботи _____ Аушева Н. М.
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Деформація зображення є актуальною задачею у багатьох сферах діяльності, якість рішення якої залежить від вибору методів та засобів. З переходом комп'ютерної графіки від двовимірних графічних моделей до тривимірних, з'являється необхідність накладання існуючих растрових зображень на тривимірні об'єкти або поверхні.

Мета роботи – проаналізувати методи деформації зображення та існуючі засоби, порівняти методи морфінгу двох зображень та розробити програмний продукт для генерації послідовності деформованих зображень та анімації морфінгу, що накладається на задану поверхню.

Записка містить 53 сторінки, 17 рисунків, 4 додатки та 20 посилань.

Ключові слова: ДЕФОРМАЦІЯ ЗОБРАЖЕННЯ, РАСТРОВІ ЗОБРАЖЕННЯ, МОРФІНГ, НАНЕСЕННЯ ЗОБРАЖЕННЯ НА ПОВЕРХНЮ, МОРФІНГ НА ПОВЕРХНІ.

ABSTRACT

Deformation of the image is a widespread problem in many areas, the quality of solving which depends on the correct choice of methods and tools. Since computer graphics used to creation the three-dimensional models, there is a need to overlay existing bitmap on them.

The purpose of this work is to analyze the methods and existing tools of deformation of bitmaps, compare the methods of the morphing and develop an application for generating a sequence of deformed images and a morphing animation superimposed on the surface.

The note contains 53 pages, 17 figures, 4 attachments and 20 links.

Keywords: IMAGE DEFORMATION, BITMAP, MORPHING, IMAGE ON THE SURFACE, MORPHING ON THE SURFACE.

ЗМІСТ

Список термінів, скорочень та позначень 7

Вступ…… 8

1. Постановка задачі деформації растрових зображень для нанесення на поверхні .. 10

2. Існуючі рішення для деформації зображення 12

2.1 Програмні засоби 12

2.2 Методи графічних перетворень 15

2.2.1 Деформація растрового зображення 15

2.2.2 Отвори та перекриття при деформації растрових зображень 21

2.2.3 Морфінг 22

2.2.4 Методи задання поверхні 23

2.2.5 Методи проекції графічних об’єктів 26

3. Теоретичне підґрунтя методів розв’язання поставленої задачі 28

3.1 Методи морфінгу 28

3.1.1 Метод сітчастого викривлення (Mesh Warping) 28

3.1.2 Тріангуляція Делоне 29

3.2 Лінійна інтерполяція 30

3.3 Криві для задання площини зображення 31

3.4 Деформація растрового зображення для нанесення на поверхню 33

3.5 Одноточкова проекція 34

4. Обгрунтування вибору засобів реалізації 35

4.1 .Net Framework 35

4.2 Windows Foundation Presentation 36

5. Опис розробленої системи 37
5.1 Опис архітектури програмного програмної системи 37
5.2 Опис реалізації методів деформації зображення 39
5.2.1 Генерація послідовності зображень для морфінгу 39
5.2.2 Генерація послідовностей зображення для створення анімації морфінгу на поверхні Безьє 41
5.3 Графічний інтерфейс користувача 42
6. Робота користувача з програмним забезпеченням 45
6.1 Системні вимоги 45
6.2 Рекомендації щодо використання 45
Висновки… 49
Список використаних літературних джерел 51
Додаток А ..... - 1 -3
Додаток Б 55
Додаток В ..... - 1 -
Додаток Г ..... - 1 -1

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

GUI – Graphical User Interface (графічний інтерфейс користувача).

DLL – Dynamic-Link Library (динамічно під'єднана бібліотека – бібліотеки, реалізовані компанією Microsoft, що являються загальними бібліотеками для операційних систем Windows).

MVP – Model-View-Presenter (архітектурний шаблон «Модель-Представлення-Представник»).

Metamorphosis – морфінг (тип анімації переходу одного зображення в інше).

WPF – Windows Presentation Foundation (технологія реалізована компанією Microsoft для створення графічного інтерфейсу користувача).

XAML – eXtensible Application Markup Language (спеціальна мова розмітки, що використовується в технології WPF).

VS – Visual Studio (середовище програмування розроблене компанією Microsoft для платформи .Net).

WinForms – Windows Forms (інтерфейс програмування програмних додатків, що відповідає за GUI і є частиною Microsoft .Net Framework).

DirectX – це набір функцій API, розроблених компанією Microsoft для простого вирішення завдань, пов'язаних з роботою комп'ютерної графіки та оптимізації продуктивності за рахунок перенесення навантаження на відеокарту комп'ютера.

ВСТУП

У сучасному світі з розвитком нових технологій та зі збільшенням доступності гаджетів зростає необхідність у якісному відображенні даних та результатів обчислень. Більшість новітніх технологій тісно пов’язані з мультимедійною сферою, оскільки дані будь-яких розрахунків потрібно подати в задовільному для користувача вигляді. Засоби візуалізації даних можуть бути різних типів: просто в текстовому форматі, побудований графік, зображення, анімація тощо. Найчастіше комбінують декілька типів візуалізації для більш зрозумілого представлення даних.

Зображення та анімація одні з найдавніших і найпоширеніших засобів поширення інформації. Вони тісно пов’язані між собою і широко використовуються майже у всіх сферах діяльності людини.

Деформація зображення є невід’ємною частиною в багатьох індустріальних сферах, таких як рекламна індустрія, кінематограф, розробка комп’ютерних ігор тощо. В зв’язку з переходом від 2D графіки до 3D однією із найскладніших та найпоширеніших завдань є деформація растрових зображень для нанесення на поверхні. Гарним прикладом такої постановки задачі – це нанесення текстур на 3D моделі при створенні комп’ютерних ігор.

Оскільки зображення має статичний стан, то для відображення зміни цього стану використовують послідовність зображень, або інакше кажучи анімацію. Для вирішення задачі відображення поступового переходу одного зображення в інше досить часто постає завдання створення морфінгу двох зображень.

Мета даної дипломної роботи дослідити методи деформації растрових зображень для нанесення їх на поверхні, дослідити та порівняти існуючі методи морфінгу зображення та створити програмний продукт за допомогою якого, користувач матиме можливість побудувати необхідну для нього поверхню та нанести на неї анімацію морфінгу двох зображень.

Актуальність вирішення описаних завдань полягає у досліджені залежності якості та швидкості генерації послідовності зображень для морфінгу від методу деформації растрових зображень.

З іншого боку, для подальшого використання морфінгу існує необхідність в нанесені цієї анімації на задану поверхню. Таке завдання є актуальним для користувачів, що працюють в рекламній індустрії або розробниками комп'ютерних ігор. Тому для вирішення поставленої задачі було розроблено програмну систему, що дозволяє нанести згенеровану анімацію морфінгу двох зображень на поверхню.

На сьогоднішній день схожих програмних систем не існує. Програмні продукти, що наразі доступні користувачам дозволяють або окремо відтворити морфінг, або окремо побудувати 3D модель та нанести на її поверхню зображення.

Зміст розділів пояснювальної записки наступний:

Перший розділ описує постановку задачі, її мету, об'єкт і предмет дослідження, проблеми, які вирішуються реалізованим програмним забезпеченням.

У другому розділі проведено огляд існуючих засобів та методів розв'язання поставленої задачі.

Третій розділ містить опис та обґрунтування вибраних для реалізації методів перетворення зображення, задання площини, морфінгу, задання поверхні та деформації зображення для нанесення на поверхню. Надає теоретичну основу для створення алгоритмічної бази.

Четвертий розділ описує перелік обраних засобів програмної реалізації та обґрунтування, чому саме такі засоби були вибрані для вирішення поставленої задачі.

П'ятий розділ містить опис програмної реалізації системи, зокрема опис реалізованих алгоритмів, складових програмних модулів та їх взаємодію.

У шостому розділі пояснюється методика роботи користувача з програмним продуктом та описується приклад сценаріїв розв'язання задач користувача.

Потенційними користувачами розробленої програмної системи можуть бути професійні фотографи, розробники комп'ютерних ігор, студенти тощо. Дана програма може бути використана для вирішення задачі створення необхідної анімації морфінгу на поверхні.

1. ПОСТАНОВКА ЗАДАЧІ ДЕФОРМАЦІЇ РАСТРОВИХ ЗОБРАЖЕНЬ ДЛЯ НАНЕСЕННЯ НА ПОВЕРХНІ

Метою дипломної роботи є розробка програмного забезпечення для дослідження деформації растрових зображень та нанесення їх на поверхні, дослідження та порівняння існуючих методів морфінгу зображення, а також створення алгоритмічної бази для перенесення анімації морфінгу на поверхню.

Об’єктом дослідження є процеси генерації послідовності зображення для морфінгу, моделювання поверхонь Безьє другого порядку та нанесення анімації на задану поверхню.

Предметом дослідження є комп’ютерні технології для реалізації процесів деформації зображення для нанесення на поверхні.

Для розв’язання поставленої задачі, були сформовані наступні завдання:

- проаналізувати існуюче програмне забезпечення та методи для деформації зображення;
- обрати засоби та методи для програмної реалізації;
- створити структуру програмного продукту;
- створити алгоритмічну базу для реалізації деформації зображення вибраними засобами та методами;
- розробити програмний продукт для створення анімації морфінгу та перенесення її на поверхню.

Завдання, що описані вище, будуть розв’язані на основі методів аналітичної геометрії та методів деформації зображень.

Для виконання поставленої задачі, необхідно розробити додаток для генерації послідовності симетричних деформованих зображень для створення анімації морфінгу та нанесення її на поверхню.

Для роботи програмною системи необхідна наступна вхідна інформація:

- контрольні точки для кожного зображення, що виділяють однотипні частини цього зображення;
- координати опорних вершин для задання поверхні;
- вибраний метод морфінгу.

Вихідною інформацією вважаються:

- побудована за контрольними точками сітка для кожного зображення для вибраного методу морфінгу;
- побудована за координатами опорних вершин модель порції поверхні для демонстрації попереднього вигляду заданої робочої поверхні;
- згенерована послідовність деформованих зображень для морфінгу;
- створена анімація морфінгу;
- створена анімація морфінгу нанесена на задану поверхню.

Програмна система повинна вирішувати наступні задачі:

1. Генерація растрових зображень деформованих відповідно до вказаного користувачем методу та сітки, вузлами якої є контрольні точки задані користувачем.
2. Демонстрація анімації морфінгу двох зображень.
3. Побудова поверхні за заданими користувачем контрольними точками та демонстрація її вигляду в просторі.
4. Накладання анімації морфінгу двох зображень на поверхню, задану користувачем, та демонстрація кінцевого результату з можливістю зміни просторового кута.

Програмний продукт розроблено мовою програмування C# під платформу .Net в середовищі Visual Studio 2019 з використанням технології WPF та спеціальної мови розмітки XAML для створення графічного інтерфейсу користувача.

2. ІСНУЮЧІ РІШЕННЯ ДЛЯ ДЕФОРМАЦІЇ ЗОБРАЖЕННЯ

Оскільки перетворення зображення є поширеним завдання для спеціалістів багатьох галузей, існує багато методів та засобів для вирішення цієї задачі. В даному розділі описані існуючі програмні рішення для роботи з зображеннями та методи трансформації.

2.1 Програмні засоби

Для деформації зображення реалізовано безліч програмних засобів, що дозволяють не тільки виконувати прості операції перетворення, а також створювати анімації, переходи, будувати тривимірні графічні об’єкти, накладати текстури тощо. Розглянемо деякі програмні додатки для перетворення зображень.

Найбільш простим та поширеним є програма Paint. Paint вбудований в операційну систему Windows і надає користувачеві інтерфейс для виконання найпростіших графічних операцій. Такими операціями є поворот зображення, віддзеркалення, можливість вирізати виділену частину, намалювати графічні об’єкти тощо. Перевагами даного програмного продукту є простота використання та невеликі системні вимоги. Недоліки полягають у неможливості виконання складних операцій та деформацій [1].

Гарним прикладом програми з розширеними можливостями є Photoshop. Photoshop надає можливість виконувати не тільки примітивні графічні операції перетворення, а також має засоби для роботи з тривимірними моделями, накладання текстур, створення анімації тощо. Даний програмний продукт широко використовуються аніматорами, фотографами та іншими спеціалістами.

Перевагами цього програмного додатку є надання широкого функціоналу для деформації та обробки зображень і графічних моделей. Недоліками програми є складність використання [2].

Іншим потужним програмним засобом є 3DS MAX, що найчастіше використовується архітекторами, дизайнерами та розробниками комп'ютерних ігор. 3DS MAX – програмне забезпечення для 3D моделювання та візуалізації дизайну, ігор та анімації (Рисунок 2.1). 3DS MAX є широко розповсюдженим професійним програмним забезпечення для створення візуалізації просторових об'єктів [3].

Переваги та недоліки цього програмного продукту схожі з Photoshop: 3DS MAX надає користувачеві потужний функціонал, проте має досить великі програмні вимоги та не є простим у використанні.

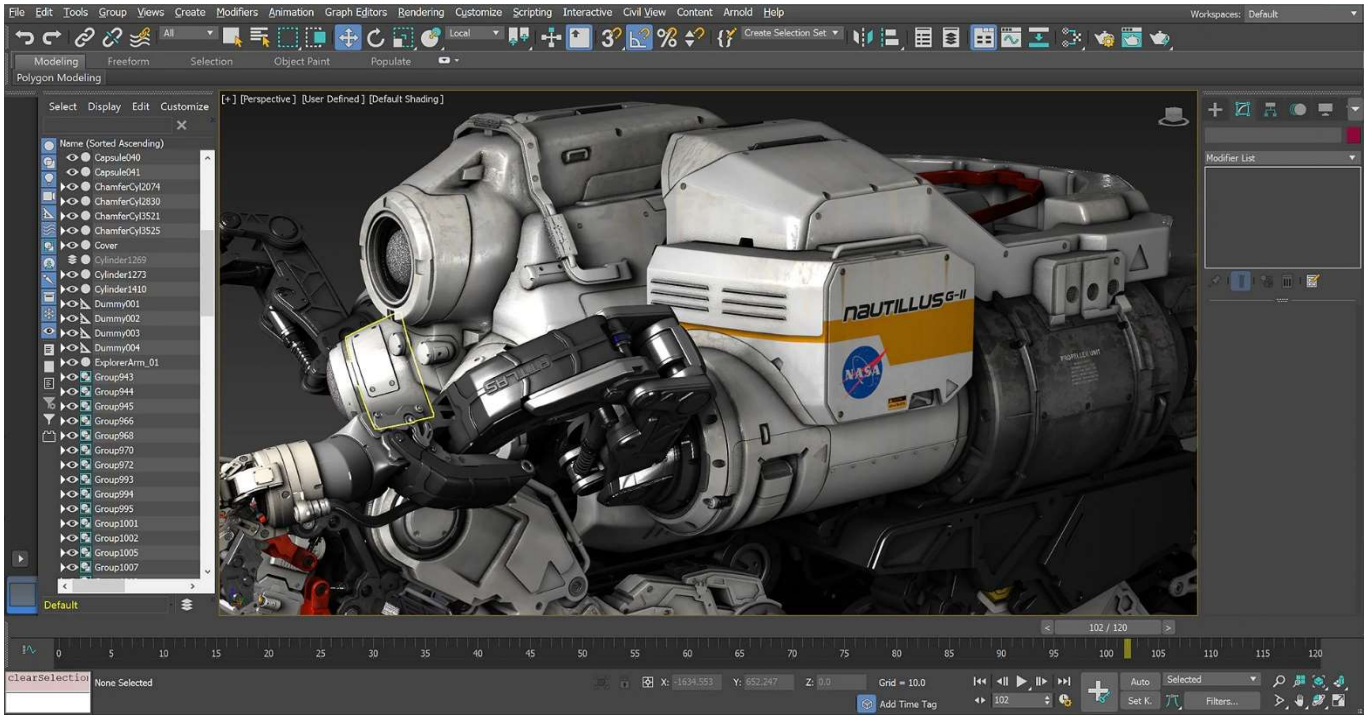

Рисунок 2.1 – Приклад робочого вікна програми 3DS MAX

Для створення анімації морфінгу двох зображень досить часто використовують Sqirlz Morph. Sqirlz Morph – це простий програмний засіб, що дозволяє перетворювати декілька зображень або відео в одне за допомогою алгоритмів морфінгу (Рисунок 2.2).

Процес перетворення зображень або відеокліпів за допомогою програмного середовища Sqirlz Morph досить простий. Необхідно просто визначити два зображення, або кліпи, які потрібно перетворити, позначити контрольні точки переходу та вибрати формат, в якому зберегти остаточну анімацію [4].

Перевагами цього програмного забезпечення в порівнянні з розробленим полягає у тому, що можна експортувати згенеровану анімацію у декількох відео форматах. Недоліком є неможливість вибрати метод морфінгу.

Рисунок 2.2 – Приклад робочого вікна програми Sqirlz Morph

Проаналізувавши існуючі програмні системи, можна помітити, що вирішення задачі створення анімації морфінгу та нанесення її на поверхню не можливе за допомогою програми. Проте схожий результат можна отримати за допомогою комбінації вихідних даних виконання декількох програмних продуктів, таких як Sqirlz Morph та 3DS MAX.

За допомогою програмного додатку Sqirlz Morph можна згенерувати морфінгу двох зображень. Однак дана система не надає можливості роботи з тривимірними моделями або поверхнями.

Для нанесення згенерованої послідовності зображень на поверхню можна використати 3DS MAX, отримавши таким чином анімації морфінгу на поверхні.

Оскільки, за допомогою одного програмного засобу вирішити поставлені завдання неможливо, в даній роботі було розроблено програмний продукт, що надає користувачеві інтерфейс, за допомогою якого він може одночасно згенерувати послідовність зображень для морфінгу, відобразити результат генерації у вигляді анімації на екрані, а також побудувати необхідну порцію поверхні та нанести на неї створену анімацію.

2.2 Методи графічних перетворень

Для перетворення зображення та графічних об’єктів використовують різні методи деформації, морфінгу, нанесення на поверхні та методи проекції багатовимірних об’єктів.

2.2.1 Деформація растрового зображення

Деформація зображення (Image warping) – це процес зміни позиції пікселя за допомогою геометричної трансформація растрового зображення. Існує багато типів деформації зображення починаючи від повороту та збільшення до нелінійної трансформації [5].

При обробці зображень деформації зображень зазвичай використовуються для усунення спотворень, а в комп’ютерній графіці – для спотворення початкового зображення. Наприклад у морфінгу зображень деформацію використовують для співставлення ключових особливостей двох зображень, що трансформуються. Найчастіше ключовими особливостями є окремі однотипні частини зображення, наприклад очей, рота і носа.

Деформація зображення, або просторове перетворення (spatial transformation), визначає співвідношення між кожною з точок вхідного зображення з кожною точкою вихідного зображення (Рисунок 2.3) [6].

Загальну функцію трансформації можна задати наступним чином:

$$[x, y] = [X(u, v), Y(u, v)], \quad (2.1)$$

де $[u, v]$ – координати початкового зображення, а $[x, y]$ – координати кінцевого зображення. Функції X та Y рівності (2.1) визначають співвідношення між кожними пікселями двох зображень [5].

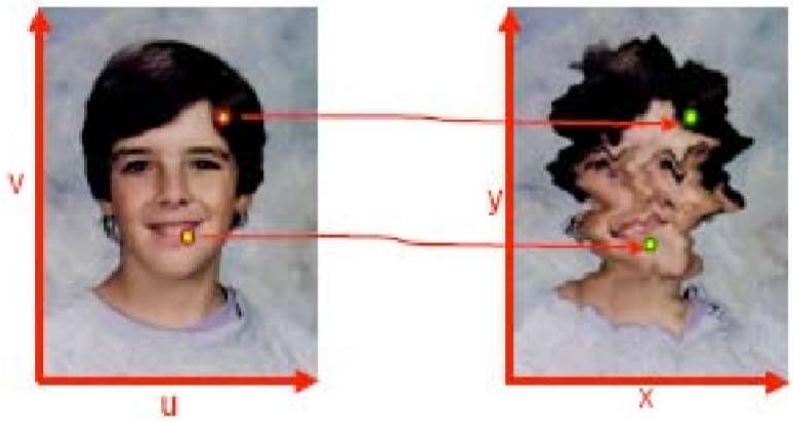

Рисунок 2.3 – Співвідношення координат точок двох зображень при деформації

Об'єкт, що визначається однорідними координатами не змінюється при множенні всіх координат на одне і те ж число відмінне від нуля. Для відображення евклідової геометрії використовують декартову систему координат, а для проективної геометрії використовують однорідні координати [5].

Однорідне представлення точок використовується для забезпечення скінченного відображення графічних об’єктів афінної та проективної геометрії. Тобто, координати нескінченно віддаленої точки можуть бути представлені за допомогою кінцевих координат.

В основі методу однорідних координат лежить подання про те, що кожна точка в n вимірному просторі може розглядатися як проекція точки з $n+1$ вимірного простору.

В порівнянні з евклідовою геометрією, де точка в реальній площині R^2 представлена векторами виду $[x, y]$, проективна має справу з проективною площиною, однорідними координатами якої є $[x', y', w]$.

В проективній геометрії вектор на площині з координатами $p = [x, y]$ буде відображений в однорідних координатах як вектор $p_h = [x', y', 1] = [xw', yw', w']$, де $w' \neq 0$. Щоб відновити фактичні координати вектору p з однорідного вектора p_h необхідно просто поділити на однорідну складову w'.

Аналогічно можна виконати перетворення в однорідні координати тривимірного вектору. У такому разі, однорідними координатами фактичного вектора $p = [x, y, z]$ вектор $p_h = [x', y', z', 1] = [xw', yw', zw', w']$.

Біраціональні перетворення задаються дробово-раціональними функціями:

$$x' = \frac{f_1(x, y)}{f_3(x, y)}$$
$$y' = \frac{f_2(x, y)}{f_3(x, y)} \qquad (2.2)$$

де f_1, f_2 та f_3 алгебраїчні багаточлени *n*-го порядку.

У біраціональних перетвореннях у відповідність ставляться криві різних порядків, але того самого роду. Рід кривої визначається як різниця між числом можливих подвійних точок і фактично існуючих [7].

Проективні перетворення задаються рівнянням (2.2), якщо f_1, f_2 та f_3 – багаточлени першого ступеня $a_1x + b_1y + c$. Таким чином для *n* вимірного простору проективні перетворення можна задати так:

$$\begin{cases} y_1 = \dfrac{a_{00} + a_{10}x_1 + \cdots + a_{n0}x_n}{a_{0n} + a_{1n}x_1 + \cdots + a_{nn}x_n} \\ \\ \cdots \\ \\ y_n = \dfrac{a_{0(n-1)} + a_{1(n-1)}x_1 + \cdots + a_{n(n-1)}x_n}{a_{0n} + a_{1n}x_1 + \cdots + a_{nn}x_n} \end{cases}$$

x_1, $x_2\ldots$ – координати точки вхідного об’єкту, y_1, $y_2\ldots$ – координати точок вихідного об’єкту [7].

Проективні перетворення мають наступні інварянти:

- Порядок – пряма перетворюється в пряму того ж порядку.
- Складне відношення 4 точок:

$$\frac{AC}{BC}:\frac{AD}{BD}=\frac{A'C'}{B'C'}:\frac{A'D'}{B'D'}$$

Проективні перетворення у векторно-параметричному вигляді задаються наступним чином:

$$r'=\frac{r_0W_0+r_xW_xx+r_yW_yy+r_zW_zz}{W_0+W_xx+W_yy+W_zz} \tag{2.3}$$

при цьому:

- $r_0(x_0,y_0,z_0)$, W_0 – початок нової системи координат;
- $r_x(x_x,y_x,z_x)$, W_x, $r_y(x_y,y_y,z_y)$, W_y та $r_z(x_z,y_z,z_z)$, W_z – координати кінців векторів осей відповідно *x*, *y* та *z* з вагами W_x , W_y та W_z;
- $r(x,y,z)$ – координати початкової області;
- $r(x',y',z')$ – координати перетвореної області.

Для зручності виконання розрахунків, можна ввести матрицю перетворення в однорідній системі координат:

$$T=\begin{bmatrix} x_xW_x & y_xW_x & z_xW_x & W_x \\ x_yW_y & y_yW_y & z_yW_y & W_y \\ x_zW_z & y_zW_z & z_zW_z & W_z \\ x_0W_0 & y_0W_0 & z_0W_0 & W_0 \end{bmatrix}$$

За допомогою векторно-параметричного, або матричного вигляду можна задати проективне перетворення для трансформації растрового зображення [7].

Афінні перетворення на площині задаються лінійними функціями. Основними інварянтами, окрім тих, що успадкували від проективних перетворень є паралельність та просте відношення 3 точок:

$$\frac{AC}{BC}=\frac{A'C'}{B'C'} => (A,B,C)=(A',B',C')$$

Афінні перетворення у векторно-параметричному вигляді:

$$r'=r_0+r_xx+r_yy+r_zz$$

при цьому:

- $r_0(x_0,y_0,z_0)$ – початок нової системи координат;
- $r_x(x_x,y_x,z_x)$, $r_y(x_y,y_y,z_y)$ та $r_z(x_z,y_z,z_z)$ – координати кінців векторів осей відповідно *x*, *y* та *z*;

- $r(x, y, z)$ – координати початкової області;
- $r(x', y', z')$ – координати перетвореної області.

У матричному вигляді в однорідних системах координат афінні перетворення можна задати наступним чином:

$$T = \begin{bmatrix} x_x & y_x & z_x & 0 \\ x_y & y_y & z_y & 0 \\ x_z & y_z & z_z & 0 \\ x_0 & y_0 & z_0 & 1 \end{bmatrix}$$

Для задання афінних перетворень зручно використовувати матричний вигляд, або просто заміть функцій X та Y у формулу (2.1) можна підставити векторно-параметричне рівняння [7].

Евклідові перетворення складають підгрупу афінних перетворень. Для того, щоб афінні перетворення стали евклідовими необхідно ввести додаткові умови:

1. Умова одиничності:

$$\begin{cases} x_x^2 + y_x^2 + z_x^2 = 1 \\ x_y^2 + y_y^2 + z_y^2 = 1 \\ x_z^2 + y_z^2 + z_z^2 = 1 \end{cases}$$

2. Умова ортогональності:

$$\begin{cases} x_x x_y + y_x y_y + z_x z_y = 0 \\ x_x x_z + y_x y_z + z_x z_z = 0 \\ x_y x_z + y_y y_z + z_y z_z = 0 \end{cases}$$

Основний інварянт евклідових перетворень – збереження відстаней між 2 точками. В наслідок цього, зберігаються також величини кутів [7].

До евклідових перетворень відносять зсув та обертання. Зсув можна задати у векторно-параметричному вигляді:

$$r' = r + r_t$$

де $r(x, y, z)$ – координати точки до перетворень, $r'(x', y', z')$ - координати точки перетвореного об’єкту, а $r_t(m, n, l)$ – вектор зсуву [8].

Зсув також можна задати у вигляді матриці однорідних координат:

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ m & n & l & 1 \end{bmatrix}$$

Обертання об’єктів залежить від того, навколо чого об’єкт необхідно перетворити. В залежності від вибраного орієнтира, можна скласти параметричне рівняння перетворення, або матрицю однорідних координат.

Наприклад, для обертання навколо осі O_z на кут Q перетворення можна задати в параметричному вигляді наступним чином:

$$\begin{cases} x' = x\cos(Q) - y\,sin(Q) \\ y' = x\sin(Q) + y\cos(Q) \\ z' = z \end{cases}$$

Також обертання можна задати в матричному вигляді. Обертання навколо осі O_z на кут Q має вигляд:

$$T_z = \begin{bmatrix} \cos(Q) & \sin(Q) & 0 & 0 \\ -\sin(Q) & \cos(Q) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Матриця обертання навколо осі O_x на кут Q має вигляд:

$$T_z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(Q) & \sin(Q) & 0 \\ 0 & -\sin(Q) & \cos(Q) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Матрицю обертання навколо осі O_y на кут Q можна задати таким чином:

$$T_z = \begin{bmatrix} \cos(\mathrm{Q}) & 0 & -\sin(Q) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(Q) & 0 & \cos(Q) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Таким чином, для того, щоб одержати 3 вимірне обертання необхідно виконати обертання навколо 3 осей координат [7]:

$$X * T = X * T_z * T_y * T_x$$

Найчастіше обертання навколо осей використовують для демонстрації проекції об’єкту. З іншого боку, для деформації зображення, або її окремої частинки необхідно виконати обертання навколо довільної точки.

Фактично, таке перетворення складається з двох кроків – потрібно спочатку пересунути об’єкт на цю точку, потім обернути на вказаний кут та по відношенню до вказаної осі координат та пересунути об’єкт назад на початкове положення. Тобто обертання навколо точки $T(m,n)$ на кут Q можна задати в матричному вигляді наступним чином [7]:

$$T = \begin{bmatrix} \cos(Q) & \sin(Q) & 0 \\ -\sin(Q) & \cos(Q) & 0 \\ -m(\cos(Q)-1)+n\sin(Q) & -n(\cos(Q)-1)-m\sin(Q) & 1 \end{bmatrix}$$

Отже, деформація растрового зображення – це процес копіювання значення пікселя із початкової координати в кінцеву, таким чином утворюючи нове зображення.

Для отримання кінцевої координати з початкової широко використовують геометричні перетворення різних типів в залежності від необхідної деформації.

Процес деформації растрового зображення може призвести до того, що нове зображення буде містити деякі артефакти, такі як отвори та перекриття.

2.2.2 Отвори та перекриття при деформації растрових зображень

Деформація растрового зображення полягає в тому, що кожен піксель початкового зображення копіюється у відповідне положення кінцевого зображення, яке визначається функціями деформації *X* та *Y*.

Деформація є простою, якщо пікселі можна розглядати як окремі точки. Якщо пікселі розглядаються як кінцеві елементи, що лежать на цілій дискретній решітці, то деформація є складною. У другому випадку виникають наступні дві проблеми: отвори та перекриття [6].

Отвори – точки, що мають невизначений піксель. Вони виникають у випадках, коли деякі значення пікселів вхідної області не покривають всієї вихідної області, або потрапляють у невизначені координати. Перекриття – точки вихідної області, в які потрапило значення двох або більше пікселів, замість одного, таким чином перекривши початкове значення (Рисунок 2.4) [5].

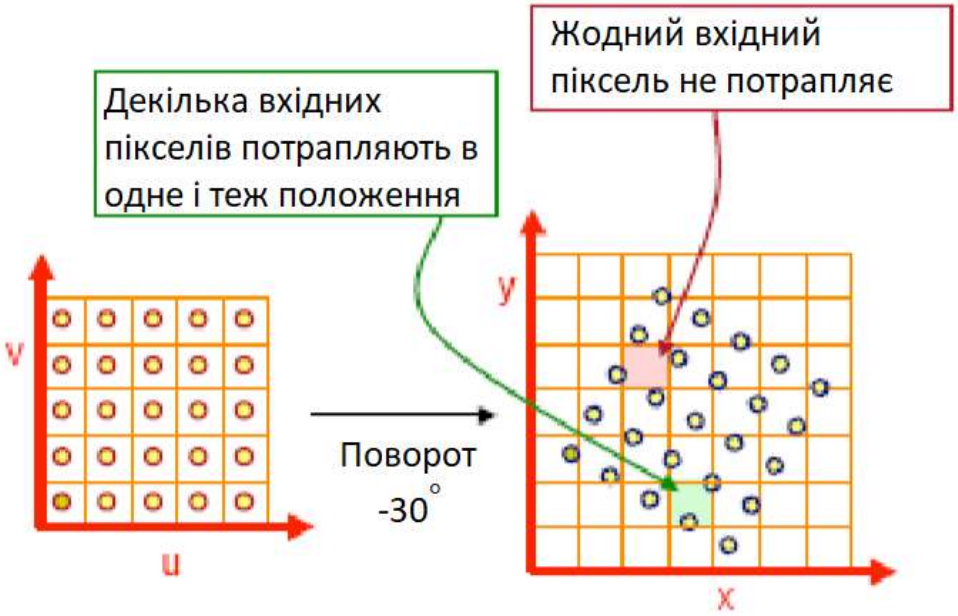

Рисунок 2.4 – Приклад утворення отворів та перекриттів при деформації зображення

Таким чином, при деформації растрових зображень можуть виникати артефакти, кількість яких залежить від типу деформації та області, що деформується.

2.2.3 Морфінг

Морфінг (Metamorphosis) – це техніка трансформації одного зображення в інше за допомогою деяких деформацій [9].

Морфінг найчастіше розглядають як технологію комп'ютерної анімації для генерації послідовності зображень, комбінація яких створює плавний перехід від початкового зображення до цільового [10]. Для генерації послідовностей деформованих зображень існує декілька алгоритмів, вибір яких залежить від необхідної якості та швидкодії методів [11].

Найпростішим методом морфінгу є метод перехресного розчинення або Cross Dissolve. Він полягає у тому, що послідовність зображень отримується шляхом зміни їх прозорості. При цьому, зміна прозорості може відбуватися на етапі відображення анімації [11].

Анімація утворюється шляхом накладання початкового зображення на кінцеве зі зміною прозорості першого. Таким чином початкове зображення поступово зникає у кінцевому, тобто початкове зображення плавно трансформується в кінцеве.

Перевагами методу cross dissolve є простота реалізацію. Також, враховуючи той факт, що деформація зображень майже не відбувається, окрім зміни прозорості, це призводить до відсутність будь-яких артефактів, тобто місць, де растрове зображення не має присвоєного значення.

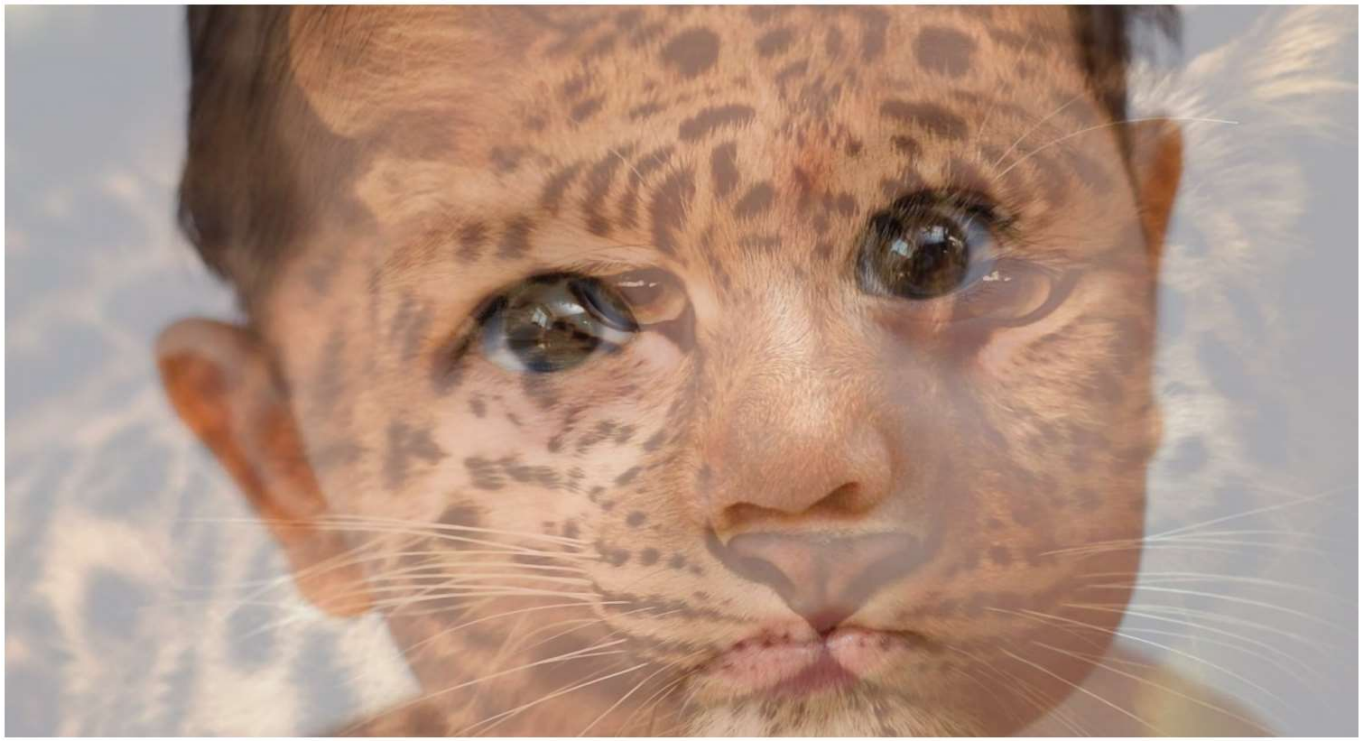

Рисунок 2.5 – Приклад морфінгу двох зображень методом Cross Dissolve

Недоліками цього методу є досить грубий перехід від початкового зображення до кінцевого. Це може бути особливо помітно, коли місцерозташування двох однотипних частин зображення, наприклад очей, не співпадають (Рисунок 2.5).

2.2.4 Методи задання поверхні

Традиційним способом представлення поверхні є використання декількох ортогональних проекцій.

Тобто, поверхня задається сіткою ортогонально-плоских кривих, що лежать на січний площині і декількома ортогональними проекціями визначених характерних просторових ліній [12].

Існує два основних методи задання поверхні.

Перший з них пов’язаний з ім’ям Кунса, коли математичну модель намагаються побудувати за вже відомими даними, а другий, пов’язаний з ім’ям Безьє, коли математичну поверхню задають з самого початку [13].

Якщо відомі чотири граничні криві $P(U,0)$, $P(U,1)$, $P(0,W)$ і $P(1,W)$ та для внутрішньої частини шматка поверхні використовується білінійна змішуюча функція, то в результаті отримуємо лінійну поверхню Кунса (Рисунок 2.6).

Лінійну поверхню Кунса можна задати наступним рівнянням [14]:

$$\begin{aligned} Q(u,w) = {} & P(u,0)(1-w) + P(u,1)w + P(0,w)(1-u) + P(1,w)\mathrm{u} \\ & - P(0,0)\,(1-u)(1-w) - P(0,1)(1-u)w \\ & - P(1,0)u(1-w) - P(1,1)uw \end{aligned} \qquad (2.4)$$

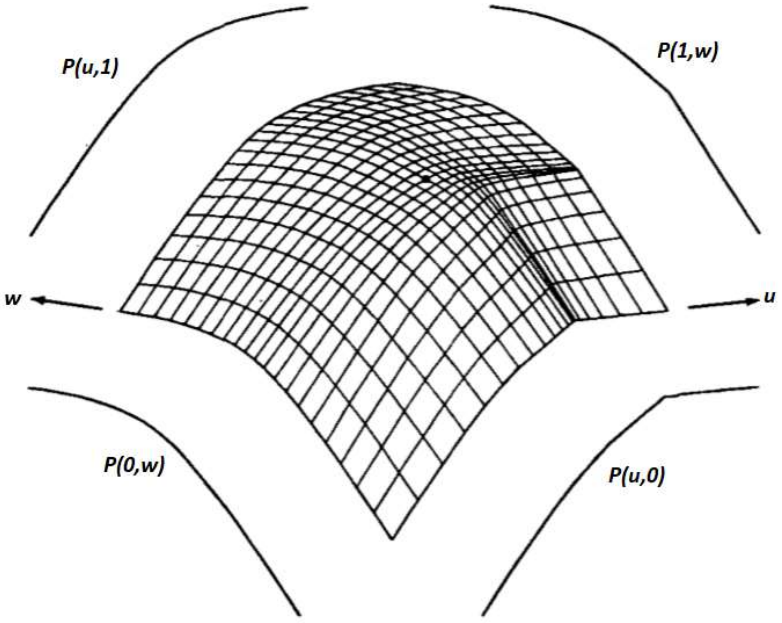

Рисунок 2.6 – Лінійна поверхня Кунса

Задати поверхню Безьє можна розширивши поняття кривих Безьє. Крива Безьє – це параметрично задана крива, у вигляді:

$$P(t) = \sum_{i=0}^{n} B_i\, J_{n,i}(t) \qquad (2.5)$$

при $0 \le t \le 1$, де базис Берштейна:

$$J_{n,i}(t) = \frac{n!}{i!\,(n-i)!}\, t^i (1-t)^{n-i}. \quad (2.6)$$

Тут n – порядок функції базису Берштейна і сегменту поліміальної кривої, на одиницю менше за кількість точок, що визначають многокутник. B_i – вершини многокутника, який визначає криву Безьє (Рисунок 2.7).

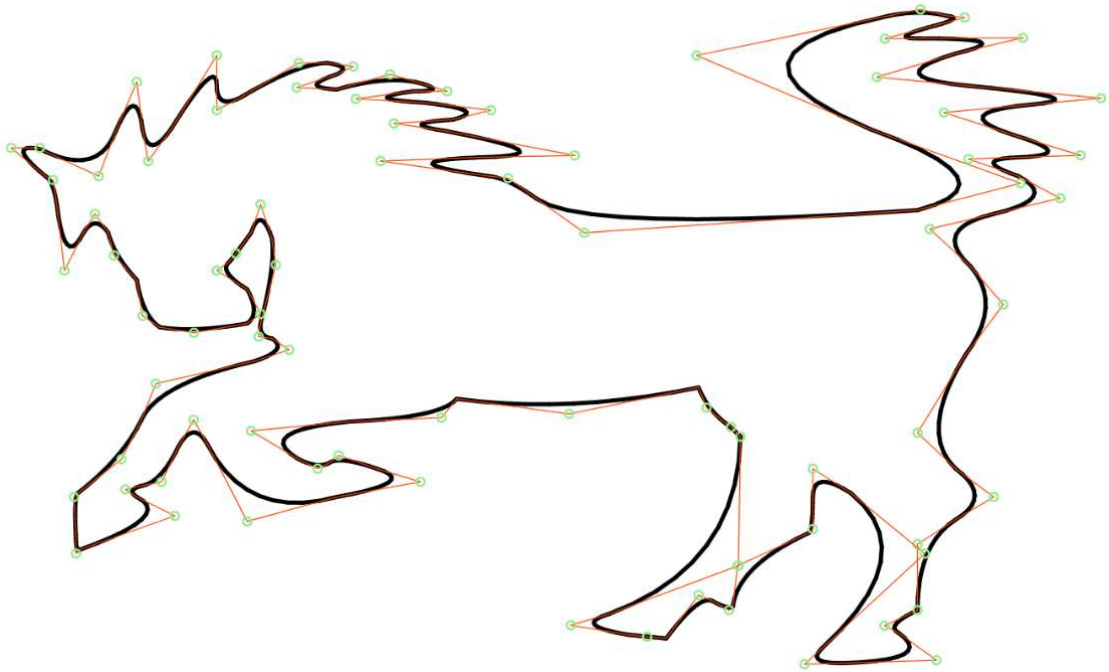

Рисунок 2.7 – Приклад побудови фігури за допомогою характеристичних трикутників кривих Безьє 2-го порядку

Поверхня Безьє в свою чергу визначається декартовим добутком кривих Безьє (Рисунок 2.8).

Декартовий добуток поверхні Безьє задається у вигляді:

$$Q(u,w) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_{i,j}\, J_{n,i}(u) K_{m,j}(w) \quad (2.7)$$

У рівнянні (2.7) функції $J_{n,i}(u)$ та $K_{m,j}(w)$ є базисними функціями в параметричних напрямках *u* та *w*.

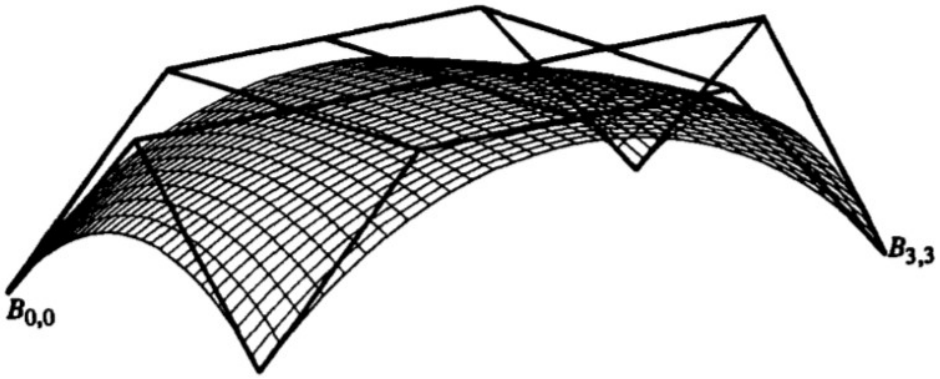

Рисунок 2.8 – Приклад поверхні Безьє

Тут $B_{i,j}$ являються вершинами полігональної сітки, що визначає поверхню [14] На Рисунок 2.8 поверхня Бізьє задана полігональною сіткою розміром 5×3.

2.2.5 Методи проекції графічних об'єктів

Оскільки екран комп'ютера – це площина, то для відображення тривимірних графічних об'єктів необхідно побудувати їх проекцію у двовимірних координатах.

В загальному випадку проекції перетворюють точки, задані в системі координат розмірністю n, в систем и координат розмірністю меншій за n. Завдання, для рішення якого найчастіше використовують проекції – це перехід від тривимірного зображення до двовимірного і навпаки [15].

Проекція тривимірного об'єкту будується за допомогою прямих проекційних променів, які називаються проекторами і які проходять через кожну точку об'єкту і, при перетині з площиною відображення, утворюють проекцію. Визначений таким чином клас проекцій називається плоскою геометричною проекцією.

Проекції діляться на два основних типи – паралельні (аксонометричні) та центральні (перспективні).

Паралельні проекції поділяються на два типи в залежності від співвідношення проектування і нормаллю до проекційної площини:

1. Ортографічні – направлення співпадають, тобто направлення проектування є нормаллю до площини проекції.
2. Косокутні – направлення проектування і нормалі до площини проекції не співпадають.

Центральна проекція будь-якого набору паралельних прямих, які не паралельні площині проекції, будуть сходитися в точці збіжності. Точок збіжності нескінченно багато. Якщо набір прямих паралельна до однієї з головних координатних осей, то їх точка збіжності називається головною точкою збіжності. Існує три види центральних проекцій в залежності від кількості точок збіжності проекції – одноточкова проекція, двоточкова проекція та триточкова проекція [15].

3. ТЕОРЕТИЧНЕ ПІДҐРУНТЯ МЕТОДІВ РОЗВ'ЯЗАННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

Для вирішення поставленої задачі були проаналізовані різні методи перетворення зображень та графічних об'єктів. В ході дослідження були визначені необхідні методи розв'язання поставленої задачі, а саме методи інтерполяції для задання площини та зміни руху контрольних точок, метод задання поверхні, методи морфінгу для порівняння та визначені способи нанесення зображення на поверхню.

3.1 Методи морфінгу

Для порівняння та створення анімації морфінгу було вибрано методи сітчастого викривлення та метод тріангуляції. За допомогою цих методів можна виконати більш плавний перехід, ніж методом перехресного розчинення, оскільки головною метою цих методів є деформація зображення для встановлення співвідношення однотипних частин зображення, наприклад очей, носа, рота тощо.

3.1.1 Метод сітчастого викривлення (Mesh Warping)

Метод сітчастого викривлення (Mesh Warping) полягає у виділені однотипних елементів зображення за допомогою контрольних точок та сіток та деформація зображення для зміни цих областей відповідно до іншого зображення [9].

На першому етапі необхідно, щоб користувач виділив за допомогою контрольних точок однотипні частини зображення. Після чого, на зображення накладається геометрична сітка. Тобто на даному етапі маємо дві сітки для початкового та кінцевого зображення.

Далі необхідно отримати послідовність сіток переходу від сітки початкового зображення в кінцеве. Ця послідовність відображає плавну деформацію зображення для співпадіння схожих частин зображення.

За допомогою будь-якого вибраного методу деформації необхідно згенерувати послідовність зображень, що будуть відповідати побудованій на попередньому етапі послідовності сіток.

Таким чином, після симетричного накладання згенерованих зображень зі зміною прозорості кожного з них отримаємо плавну анімацію переходу одного зображення в інше.

Переваги цього методу є досить проста реалізація і побудова плавного переходу за рахунок деформації зображень.

Недоліки методу полягають у тому, що при деформації частин зображень можуть з’являтися деякі артефакти. Це відбувається тому, що при деформації растрових зображень, таких як розтягнення, чи зміна кута можуть з’являтися нові координати точок зображення значення кольору яких невідомо, оскільки в початковому зображенні вони відсутні. Це може призвести до неякісного кінцевого зображення. Наявність та кількість таких артефактів залежить від вибраного методу деформації зображення під час переходу від однієї сітки до іншої.

3.1.2 Тріангуляція Делоне

Для морфінгу також досить часто використовують метод тріангуляції Делоне [11]. Тріангуляцією називається пленарний граф, всі внутрішні області якого є трикутниками [16].

Випуклою тріангуляцією називається така тріангуляція, для якої мінімальний многокутник, що охоплює всі трикутники, буде випуклим.

Тріангуляція задовольняє умовам Делоне, якщо в середину кола, описаного навколо будь-якого трикутника, не потрапляє жодна точка заданої тріангуляції.

Тріангуляція називається тріангуляцією Делоне, якщо вона є випуклою та задовольняє умовам Делоне [16].

На першому етапі морфінгу методом тріангуляції Делоне необхідно побудувати випуклу тріангуляцію, що задовольняє умовам Делоне, для заданих контрольних точок, тобто на початкове і кінцеве зображення накласти трикутну сітку [11].

Після цього, формується послідовність трикутних сіток переходу від початкового зображення до кінцевого. Далі генеруємо шляхом деформації послідовність зображень відповідно до кожної із отриманих сіток та симетрично накладаємо кожне із зображень зі зміною прозорості для створення анімації морфінгу.

Перевагами цього методу є плавний перехід від одного зображення до іншого, при цьому та невелика кількість артефактів.

Недоліками ж є складність розрахунку, наявність артефактів, кількість яких залежить від вибору методу деформації кожного наявного підзображення.

Існує і багато інших методів морфінгу, як наприклад морфінг поля, проте описані вище методи є найбільш поширеними серед систем вирішення задач морфінгу та створення анімації переходу одного зображення в інше.

3.2 Лінійна інтерполяція

Інтерполяція – спосіб знаходження в обчислювальній математичці проміжних значень функції за наявним набором відомих дискретних значень. Інтерполяція – це вид апроксимації, коли обов'язкове виконання наступної умови – проходження інтерполяційної кривої через задані дискретні точки [9].

Лінійна інтерполяція – це інтерполяція алгебраїчним двочленом наступного вигляду $P(x) = ax + b$ функції f, задана двома точками x_0 та x_1 на відрізку $[a, b]$.

Лінійну інтерполяції двох точок A та B можна задати у векторному вигляді наступним чином:

$$r' = (1 - t)r_A + t\, r_B \tag{3.1}$$

де $r_A(x_A, y_A, z_A)$ і $r_B(x_B, y_B, z_B)$ – координати точок A та B відповідно, $r'(x', y', z')$ - координати точки між A і B, а t – коефіцієнт ($0 \leq t \leq 1$) [9].

Отже, щоб отримати проміжні точки, що знаходяться на прямій AB і лежать між ними, необхідно для підставити в рівняння (3.1) значення параметра t від 0 до 1.

Метод лінійної інтерполяції було вибрано для реалізації зміни положення контрольних точок двох зображень при створенні необхідних сіток в процесі морфінгу двох зображень.

3.3 Криві для задання площини зображення

Для задання площини зображення було вирішено використовувати сплайнові криві Катмулл-Рома, а також криві Бізьє другого порядку.

Сплайнові криві Катмулл-Рома зручно використовувати для методу сітчастого викривлення під час морфінгу двох зображень, оскільки сітка, що використовується в даному методі, є чотирикутна, таким чином легко задати інтерполяцію Катмулл-Рома по 4 точкам [17].

Нехай на відрізку $[a, b]$ задана сітка w:

$$a = x_0 < x_1 < \cdots < x_{m-1} < x_m = b.$$

Точки x_0 та x_m називаються граничними вузлами сітки w, а точки x_1, …, x_{m-1} – її внутрішніми вузлами. Сітка називається рівномірною, якщо відстань між будь-якими двома сусідніми вузлами однакові.

Функція $S(x)$, задана на відрізку $[a, b]$, називається сплайном порядку p+1, якщо ця функція:

1. На кожному з відрізків $\Delta_i = [x_i, x_{i+1}]$, де i = 0, 1, …, $m - 1$, являється многочленом вказаної степені $p \geq 2$, тобто може бути записана у вигляді:

$$S(x) = S_i(x) = \sum_{k=0}^{p} a_k^{(i)} (x - x_i)^k \tag{3.2}$$

2. $p - 1$ разів неперервно диференційована на відрізку $[a, b]$, тобто виконується умова:

$$S(x) \in C^{p-1}[a, b]. \tag{3.3}$$

Індекс (i) числа у чисел $a_k^{(i)}$ вказує на те, що набір коефіцієнтів, якими визначається функція $S(x)$, на кожному виділеному відрізку Δ_i свій [18].

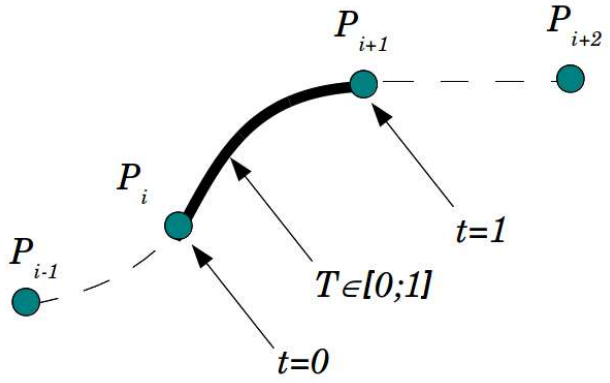

Рисунок 3.1 – Схема побудови Catmull-Rom кривої

По заданому масиву P_0, P_1, P_2, P_3 (елементарна) сплайнова крива Catmull-Rom (Рисунок 3.1) визначається за допомогою рівняння, що має наступний вигляд:

$$R(t) = \frac{1}{2} - t(1-t)^2 P_0 + (2 - 5t^2 + 3t^3)P_1 + t(1 + 4t - 3t^2)P_2 - t^2(1-t)P_3 \quad (3.4)$$

де $0 \le t \le 1$.

У матричному вигляді крива Catmull-Rom виглядає наступним чином:

$$R(t) = PMT\ ,\ 0 \le t \le 1 \quad (3.5)$$

де маємо:

$$P = [P_0 \quad P_1 \quad P_2 \quad P_3] = \begin{bmatrix} x_0 & x_1 & x_2 & x_3 \\ y_0 & y_1 & y_2 & y_3 \\ z_0 & z_1 & z_2 & z_3 \end{bmatrix}$$

$$R = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} \qquad M = \frac{1}{2}\begin{bmatrix} 1 & -3 & 3 & -1 \\ 4 & 0 & -6 & 3 \\ 1 & 3 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T = \begin{bmatrix} t^0 \\ t^1 \\ t^2 \\ t^3 \end{bmatrix}$$

Матриця M називається базисною матрицею сплайну Catmull-Rom [18].

Для методу тріангуляції морфінгу двох зображень зручно використовувати криві Безьє другого порядку, оскільки сітка, що використовується при деформації зображення є трикутна, таким чином її легко інтерпретувати в набір характеристичний трикутників.

Векторно-параметричне рівняння кривої Безьє можна задати у наступному вигляді:

$$r = r_0(1-t)^2 + 2r_1(1-t)t + r_2 t^2. \quad (3.6)$$

де $r_0(x_0, y_0, z_0)$, $r_1(x_1, y_1, z_1)$ і $r_2(x_2, y_2, z_2)$ – координати вершин характеристичного трикутника, а t – коефіцієнт ($0 \leq t \leq 1$) [13].

3.4 Деформація растрового зображення для нанесення на поверхню

. Для того, щоб нанести растрове зображення на поверхню його необхідно деформувати.

Деформація растрового зображення полягає у встановленні відповідності значення пікселя між початковим зображення і кінцевим за допомогою рівності (2.1). Саме функції X та Y визначають співвідношення між кожними пікселями двох зображень.

Функції X та Y мають задавати перехід від координати площини зображення до відповідної точки, що знаходиться на заданій поверхні. Вектор такого переходу можна задати наступним чином:

$$r'(x, y, z) = [X(u, v), Y(u, v), Z(u, v)] = Q(u, v) \quad (3.7)$$

Нехай, маємо растрове зображення яке необхідно перенести на задану порцію поверхні Безьє 2-го порядку. Поверхня задана набором вершин $B_{i,j}$ полігональної сітки, що має розмір 3×3, та рівністю (2.7).

Необхідно задати співвідношення між точкою площини зображення та точкою заданої поверхні. Оскільки поверхня Безьє 2-го роду за формулою (2.7) – це декартів добуток кривих Безьє 2-го роду, для зручності можна задати поверхню у векторному вигляді наступним чином:

$$r = r_0(1 - u)^2 + 2r_1(1 - u)u + r_2u^2, \quad (3.8)$$

де

$$r_i = r_{i0}(1 - v)^2 + 2r_{i1}(1 - v)v + r_{i2}v^2. \quad (3.9)$$

Виберемо за u напрямок по ширині зображення, тоді v – напрямок по висоті. У такому випадку, співвідношення (3.7) набуває наступного вигляду:

$$r'(x, y, z) = r(u, v), \quad (3.10)$$

де r – вектор деформації зображення для нанесення його на площину.

Оскільки відповідність між координатами точок растрового зображення та площини була встановлена, необхідно перенести значення кожного пікселя зображення у відповідну координату поверхню.

3.5 Одноточкова проекція

Для того, щоб перейти від трьох координат до двох можна скористатися одноточковою проекцією. Суть одноточкової проекції полягає в тому, що існує лише одна точка збіжності для всіх проекторів.

Наприклад, для того, щоб отримати проекцію, центр якої заходиться в точці з координатами $(0,0,k)$, точки $A(x,y,z)$, необхідно скористатися рівняннями (3.11) та (3.12).

$$x^* = \frac{x}{1-\frac{z}{k}} \tag{3.11}$$

$$y^* = \frac{y}{1-\frac{z}{k}} \tag{3.12}$$

Таким чином, використовуючи формули (3.11) та (3.12) можна отримати проекцію тривимірного графічного об’єкту та відтворити його на екрані монітору [15].

4. ОБГРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

Оскільки однією з найрозповсюджених операційних систем є Windows на платформі .Net, тому саме ця платформа була вибрана для реалізації програмного продукту деформації зображення та створення анімації морфінгу двох зображень на поверхні.

В ході написання програмного продукту було створено рішення Visual Studio Solution, що містить два C# проекти типу бібліотеки Class Library та додатку WPF Application. Для написання програмного продукту було вибрано платформу .NET Framework 4.7.2 на операційній системі Windows 10. Як середовище програмування було вибрано Microsoft Visual Studio 2019 Community. Мова програмування – об’єктно-орієнтована мова програмування C#. Для побудови графічного інтерфейсу було використано технологію Windows Presentation Foundation (WPF) та декларативна мова розмітки XAML. Даний вибір був зроблений враховуючи технології та мови програмування, що використовуються на базі практики, та з урахуванням майбутніх потреб користувачів.

4.1 .Net Framework

.Net – це безкоштовна платформа від компанії Microsoft, що є крос-платформною і відкритою для розробників програмного забезпечення, призначення якої полягає у створенні різнотипних програм. Платформа .Net дозволяє використовувати різні мови програмування (C#, F#, Visual Basic, C++ тощо) та бібліотеки для створення веб, мобільних та ігрових програмних систем [19].

.Net Framework поділяється на дві частини – середовище виконання та система розробки програмних продуктів.

Common Language Runtime (CLR) – середовище виконання, яке підходить для різних мов програмування. При компіляції програмного коду написаного мовою, що підтримується CLR, CLR-сумісні компілятори генерують IL-код (код на мові Intermediate Language). IL – це проміжна мова розроблена Microsoft, яку вбудований JIT-компілятор (Just in time compiler) перетворює в машинний код, що виконується.

Як система розробки, .Net Framework надає можливість компілювати програмний код написаний різними мовах програмування, які підтримуються CLR, в CIL. Після цього, як середовище виконання, тобто як віртуальна машина, виконати цей код за допомогою JIT [19].

Платформа .Net є дуже широко розповсюджена, оскільки вбудована в операційну систему Windows та завдяки своїм основним перевагам для розробників програмного забезпечення, а саме широкий вибір мов програмування, що підтримуються, та надання можливості розробляти програмне забезпечення різних типів, включаючи крос-платформні додатки.

4.2 Windows Foundation Presentation

Windows Foundation Presentation (WPF) – технологія, що є частиною платформи .Net і представляє собою підсистему для побудови графічних інтерфейсів. Головною відмінністю WPF від інших технологій, що застосовуються на базі платформи .Net (наприклад WinForms), є те, що відображення елементів керування інтерфейсу засновані на технології DirectX. Таким чином, значна частина навантаження з відображення графічних об’єктів на моніторі лягає на графічний процесор відеокарти комп’ютера, що дозволяє прискорити роботу програмного забезпечення [20].

Окрім цього, головними перевагами цієї технології полягають у тому, що вона підтримує мову програмування C#, можливість декларативного визначення графічного інтерфейсу за допомогою спеціальної мови розмітки XAML та інші переваги, такі як автоматичне масштабування для різних розмірів робочого вікна.

5. ОПИС РОЗРОБЛЕНОЇ СИСТЕМИ

Розроблене програмне забезпечення складається з двох проектів. Перший з проектів є бібліотекою .Net Framework GeneralModelLibrary, а другий проект – ImageWarpingApp.

Розробка системи поділялась на два етапи:

1. Розробка бібліотеки, що містить всю бізнес логіку, всі необхідні геометричні операції, розрахунки та реалізації методів вирішення поставленої задачі.
2. Розробка графічного інтерфейсу програми.

Після виконання всіх необхідних дій для налагодження роботи графічного інтерфейсу та бібліотеки геометричних розрахунків, був скомпільований кінцевий програмний продукт, який може бути запущений користувачем на його комп’ютері.

5.1 Опис архітектури програмного програмної системи

Програмний продукт містить в собі дві компоненти – бібліотеку GeneralModelLibrary.dll та виконуваний файл ImageWarpingApp.exe. Діаграма компонентів має вигляд (Рисунок 5.1):

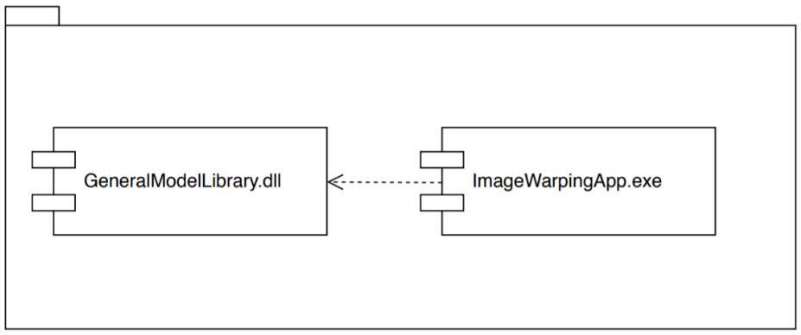

Рисунок 5.1 – Діаграма компонентів розробленого програмного забезпечення

Структура програмного забезпечення показана на Рисунок 5.2.

Рисунок 5.2 – Структура розробленого програмного забезпечення

Бібліотека розрахунків GeneralModelLibrary.dll містить всю необхідну логіку для генерації послідовності зображень двома методами морфінгу, а саме методом сітчастого викривлення та тріангуляції. Таким чином, розроблену бібліотеку можна використовувати незалежно від інтерфейсу користувача та надає можливість підміняти графічний інтерфейс в процесі подальших розробок.

Графічний інтерфейс в свою чергу надає всі необхідні програмні засоби управління для вирішення задачі морфінгу на поверхні.

5.2 Опис реалізації методів деформації зображення

Головною метою розробленої бібліотеки є надання користувачеві необхідні класи та методи для вирішення поставлених геометричних операцій, а саме:

- генерація послідовностей зображення для морфінгу для методів mesh morphing та тріангуляції;
- побудова поверхні Безьє другого порядку;
- накладання зображення на побудовану поверхню;
- генерація послідовностей зображення для створення анімації морфінгу на поверхні Безьє
- надання розширювальних методів для обчислення деяких загальних геометричних операції (наприклад находження центру кола трикутника чи відстані від точки до прямої).

Головним класом, за допомогою можна виконати необхідні розрахунки є клас MorphingModel.cs. Також були створені статичний клас з методами, що реалізують прості геометричні операції, які необхідні для подальших обчислень, та деякі класи, що представляють необхідні прості фігури, як Point, Circle тощо.

5.2.1 Генерація послідовності зображень для морфінгу

Для генерації послідовності зображень було реалізовано два методи морфінгу – метод mesh warping та тріангуляція. Два методи необхідні лише для того, щоб можна будо порівняти два алгоритми морфінгу.

Для алгоритму mesh warping було згенеровано сітки з чотирикутників для кожного зображення, що з’єднують контрольні точки (Рисунок 5.3).

Після цього генерується набір сіток переходу від сітки одного зображення до іншого за допомогою застосування методу лінійної інтерполяції (3.1) для всіх заданих контрольних точок зображення [9].

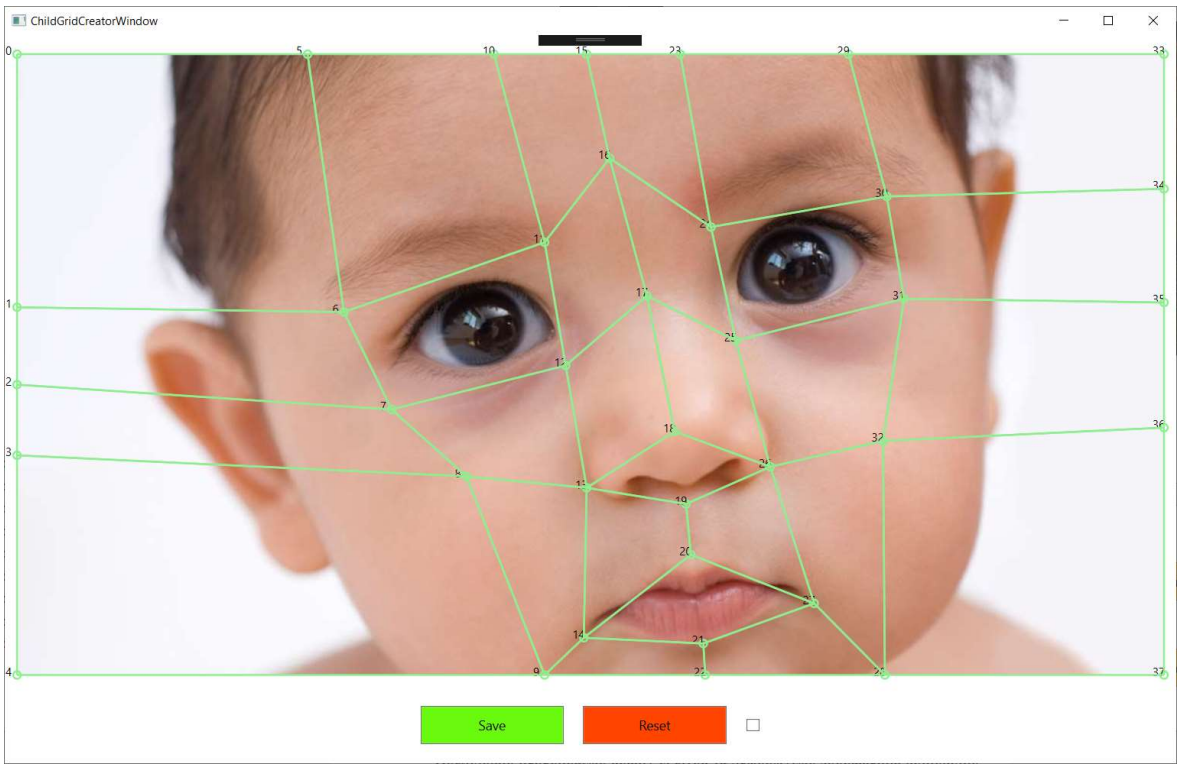

Рисунок 5.3 – Приклад візуалізації сітки для алгоритму mesh warping

Послідовно перебираємо кожну із сіток та деформуємо зображення відповідно до нової. Для задання площини кожного з чотирикутників зображення використовується метод Catmull-Rom (3.4).

Сітка розбивається на окремі чотирикутники утворюючи масив. Далі ітераційно кожен з цих полігонів обробляється за наступним алгоритмом:

1. Визначаються сусідні чотирикутники справа та зліва поточної частини зображення.

2. Виконуємо лінійну інтерполяцію зверху вниз для кожного ребра поточного та сусідніх чотирикутників. В даному випадку ми сформували таким чином рядки матриці, яка задає координати кожної точки полігону.

3. Рухаючись вниз по ребрам обчислюємо за формулою Catmull-Rom (3.4) задаємо сітку на площині для початкового і кінцевого чотирикутників [18].

4. Встановлюємо відповідність між значеннями точок в початковому полігоні і в кінцевому.

5. Додаємо отриману деформовану частинку зображення до кінцевого растрового зображення.

За допомогою такого алгоритму буде згенерована деяка послідовність зображень переходу від початкової сітки до кінцевої для двох зображень морфінгу. Після чого, для відображення анімації морфінгу ці зображення зчитуються та накладаються один на одного зі зміною прозорості кожного зображення.

Для алгоритму тріангуляції застосовується схожий за послідовністю дій, але різний за розрахунками, алгоритм.

Після того, як створюємо масив трикутників із трикутної сітки для кожного з зображень, генерується нова послідовність сіток за допомогою лінійної інтерполяції (3.1) між контрольними точками.

Послідовно перебираємо кожну із сіток та деформуємо зображення відповідно до нової сітки. Для задання площини кожного з трикутників зображення використовується крива векторне дробово-раціональне рівняння (2.3).

Сітка розбивається на окремі трикутники утворюючи масив. Далі проходимо по цьому масиву і для кожної сторони поточного трикутника виконуємо наступні дії:

1. Знаходимо координату середини сторони.
2. За допомогою лінійної інтерполяції (3.1) рухаємося від центру сторони до протилежної вершини трикутника.
3. На кожному кроці лінійної інтерполяції будуємо криву Безьє 2-го порядку (2.5) та переносимо значення з початкового трикутника зображення в кінцевий.
4. Додаємо утворену частину трикутника до кінцевого зображення

За допомогою цього алгоритму буде згенерована послідовність зображень переходу від початкової сітки до кінцевої для кожного зображення морфінгу.

Після чого, для відображення анімації морфінгу ці зображення зчитуються та накладаються один на одного зі зміною прозорості кожного зображення.

5.2.2 Генерація послідовностей зображення для створення анімації морфінгу на поверхні Безьє

Оскільки кінцевою метою є створення анімації морфінгу на поверхні, то необхідно спочатку згенерувати послідовності зображення для морфінгу одним із

вище описаних методів. Якщо послідовність зображень згенерована, то кожне з них необхідно накласти на поверхню.

Для того, щоб перенести растрове зображення на поверхню необхідно побудувати матрицю відношення значення точки на зображенні до точки на поверхні.

Нехай напрямок u – це напрямок вздовж ширини, а v – напрямок вздовж висоти. Тоді отримаємо, що для встановлення відповідності необхідно створити матрицю A розмірами $n \times m$, де n і m – ширина та висота зображення.

Далі за допомогою вектора координат (3.8) заповнюємо матрицю відповідності координат. Таким чином отримаємо співвідношення між координатами зображення та координатами на площині. Для накладання растрового зображення на поверхню необхідно взяти значення кожного пікселя зображення в точці $M(i, j)$ та присвоїти для координати поверхні, що знаходиться в $A[i, j]$ заповненого масиву за відповідними індексами.

Для того, щоб зберегти згенеровану поверхню з накладеним на неї растровим зображення необхідно спочатку спроектувати поверхню на площину. Використовуючи формули (3.11) та (3.12) можна отримати одноточкову проекцію зображення накладеного на поверхню.

Оскільки програма надає можливість користувачеві змінювати кут проекції, то для цього необхідно згенерувати декілька проекцій однієї ж поверхні обертаючи її навколо осі Oz.

5.3 Графічний інтерфейс користувача

Для створення графічного інтерфейсу було використано технологію WPF та спеціальну мову розмітки XAML. Створення інтерфейсу користувача відбувалось в два етапи: створення інтерфейсу та підготовка інтерфейсу для роботи із бібліотекою, що містить бізнес логіку програмного продукту.

За допомогою мови XAML та засобів середовища програмування Visual Studio 2019 був створений наступний графічний інтерфейс (Рисунок 5.4).

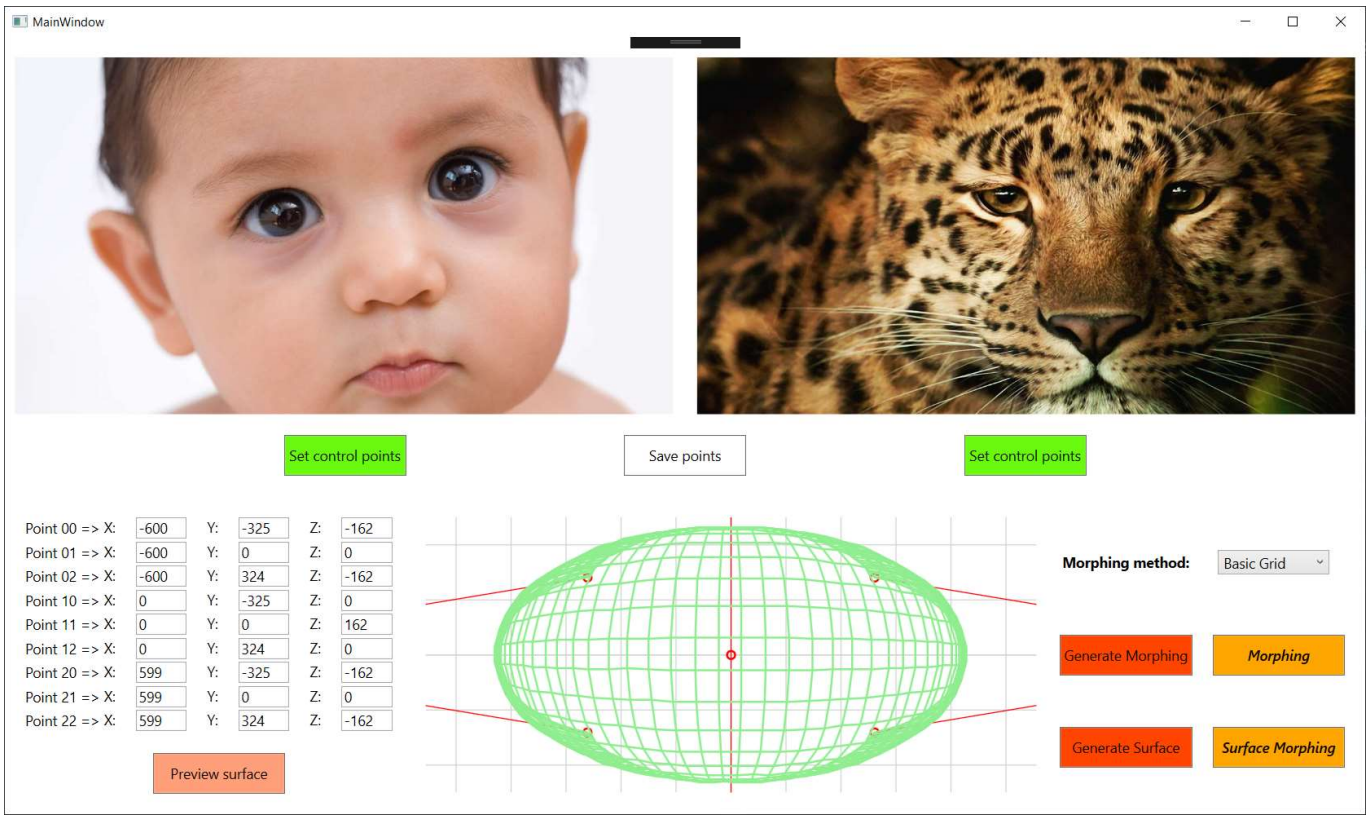

Рисунок 5.4– Приклад головного робочого вікна програми

Інтерфейс містить всі необхідні елементи контролю програми, а саме:

- приклади зображень, над якими програма наразі виконує дії;
- кнопки для встановлення контрольних точок для кожного із зображень;
- кнопка для збереження встановлених контрольних точок;
- поля вводу координат для задання поверхні та кнопка генерації прикладу зовнішнього вигляду поверхні;
- поле демонстрації заданої користувачем поверхні;
- поле для вибору методу морфінгу та кнопки для генерації послідовностей зображень та відображення анімації морфінгу.

Також було створено декілька додаткових вікон для відображення анімації морфінгу та задання контрольних точок для кожного з зображень. Завдяки технології WPF всі вікна автоматично добре масштабовані, тому підтримують різні розміри вікон за замовчуванням.

Для налагодження роботи графічного інтерфейсу з бізнес моделлю був реалізований шаблон проектування MVP. Цей шаблон був вибраний завдяки своїй

простій реалізації, можливості легко розширювати графічний інтерфейс та можливість реалізації графічного дизайну (Рисунок 5.5).

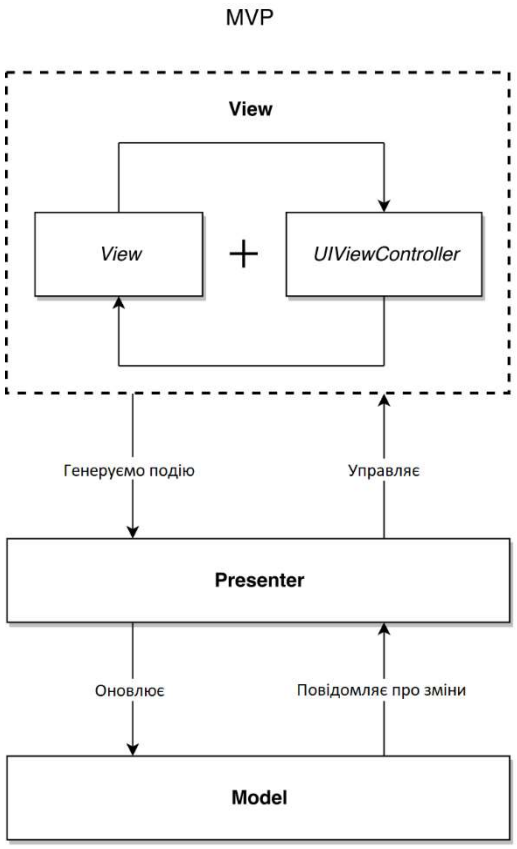

Рисунок 5.5 – Модель MVP

В даному випадку, Model – є спеціальний клас бібліотеки, що містить всю необхідну бізнес логіку для проведення геометричних обчислень.

6. РОБОТА КОРИСТУВАЧА З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

Для забезпечення безвідмовної роботи розробленої програмної системи створення анімації морфінгу та нанесення її на поверхню необхідно дотримуватися основних вимог для правильної інсталяції та рекомендацій щодо її використання.

6.1 Системні вимоги

Для роботи з розробленою програмною системою персональний комп'ютер повинен мати встановлену операційну систему Windows 10 та необхідний об'єм пам'яті на жорсткому диску, а саме не менше ніж 200 Мб вільного місця.

Для першого запуску програми необхідно мати на своєму комп'ютері файл ImageTransformationApplication.zip та розархівувати його. Оскільки попереднього встановлення програма не потребує і вона не використовує ніяких додаткових пакетів для встановлення, програма готова до роботи.

6.2 Рекомендації щодо використання

Для того, щоб запустити програму необхідно перейти в директорії, яку отримали шляхом розархівації файлу, описану в 6.1, та двічі запустити подвійним кліком лівої клавіші миші ImageWarpingApp.exe.

Після цього, перед користувачем відкриється головне робоче вікно (Рисунок 5.4), що містить в собі всі необхідні засоби управління для користування програмою. Рисунок 6.1 містить в собі окремо виділені компоненти управління програмою.

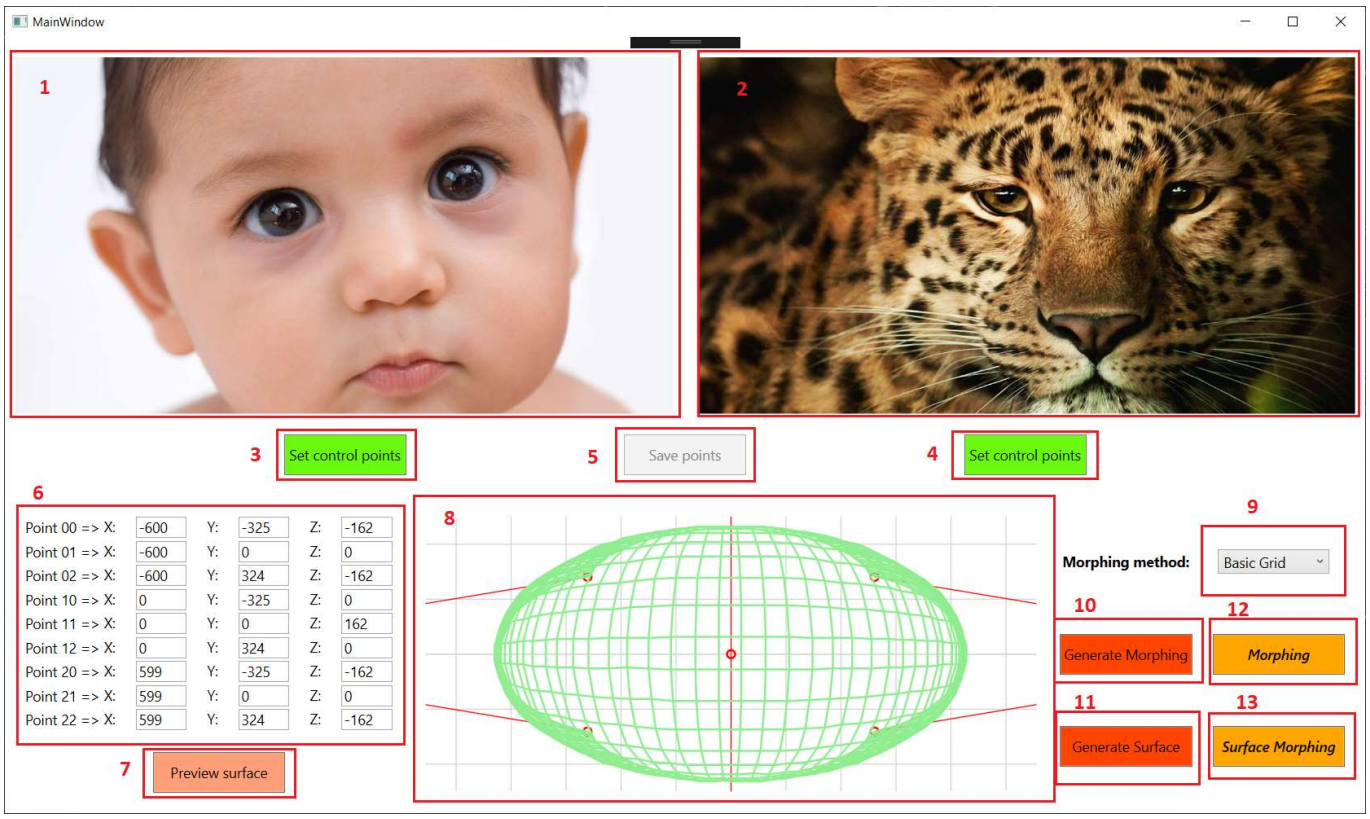

Рисунок 6.1– Головне робоче вікно програми з виділеними компонентами управління

Головне робоче вікно містить наступні компоненти (Рисунок 6.1):

1. Приклади початкового зображення, яке буде деформуватися;
2. Приклади кінцевого зображення, яке буде деформуватися;
3. Set control points – кнопка встановлення контрольних точок для початкового зображення 1;
4. Set control points – кнопка встановлення контрольних точок для кінцевого зображення 2;
5. Save points – кнопка для збереження встановлених контрольних точок;
6. Поля вводу координат для задання користувачем поверхні;
7. Preview surface – кнопка генерації прикладу зовнішнього вигляду поверхні, що буде відображений у полі 8;
8. Поле для демонстрації заданої користувачем поверхні;
9. Поле вибору методу морфінгу;
11. Generate Morphing – кнопка генерації послідовності зображень для морфінгу методом, що встановлений у полі 9;

12. Generate Surface – кнопка для генерації послідовності зображень накладеної анімації морфінгу на поверхню, що зображена на полі 8 методом, що встановлений у полі 9;

13. Morphing – кнопка, що відкриває нове вікно (рисунок 5.2) для відображення анімації морфінгу встановленим в полі 9 методом.

14. Surface Morphing – кнопка, що відкриває нове вікно (рисунок 5.3) для відображення анімації морфінгу на заданій поверхні та встановленим методом в полі 9.

Для початку роботи, необхідно встановили контрольні точки для кожного із зображень (Рисунок 5.3) та зберегти. Програма має встановлені контрольні точки за замовчуванням. Після цього, необхідно вибрати метод морфінгу в полі 9 та натиснути кнопку Generate Morphing. Доведеться почекати, поки програма виконає всі необхідні розрахунки та згенерує необхідну послідовність зображень. Для перегляду результату треба натиснути на кнопку Morphing (Рисунок 6.2).

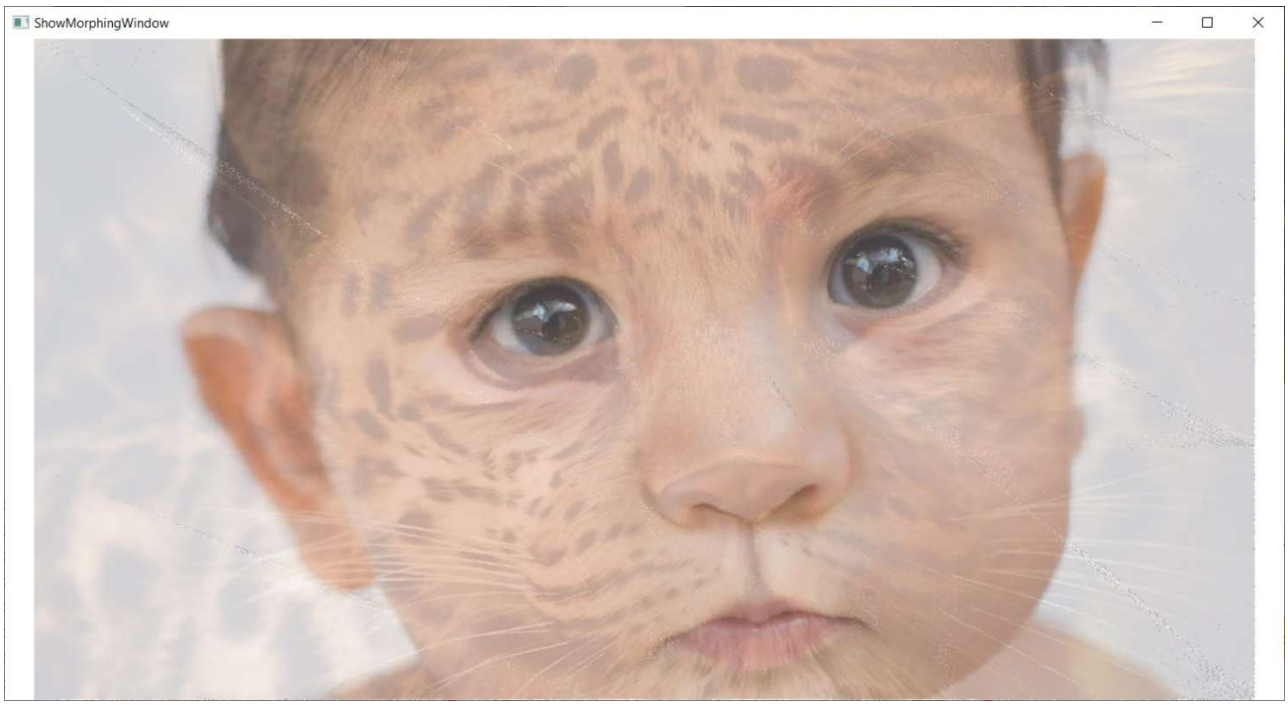

Рисунок 6.2 – Приклад відображення програмою анімації морфінгу

Для створення анімації морфінгу на поверхні необхідно задати точки у полях вводу 6. Для попереднього перегляду у полі 8 заданої поверхні треба натиснути на кнопку Preview surface.

Якщо задана поверхня влаштовує, то для генерації послідовності необхідних зображень потрібно натиснути на кнопку Generate Surface вибравши заздалегідь в полі 9 метод морфінгу. Після того, як програма закінчить розрахункові дії можна переглянути результат анімації натиснувши кнопку Surface Morphing (Рисунок 6.3).

Рисунок 6.3 – Приклад відображення програмою анімації морфінгу на заданій поверхні

У вікні, показаному на Рисунок 6.3, знизу є спеціальний повзунок, за допомогою якого можна змінювати кут проекції поверхні навколо осі Oz.

ВИСНОВКИ

Дипломну роботу присвячено актуальному напрямку деформації зображень для морфінгу та нанесення растрових зображень на поверхню. При вирішенні поставлених задач отримано наступні результати:

1. Проаналізовані існуючі програмні засоби і дійшли до висновку, що для вирішення поставленого завдання можна скористатися додатками Sqirlz Morph для отримання анімації морфінгу та 3DS MAX для нанесення растрового зображення на поверхню.
2. Досліджено методи перетворення зображення і прийнято рішення використовувати методи сітчастого викривлення та тріангуляції для морфінгу двох зображень, для задання площини зображення було вибрано метод Катмул-Рома та криві другого порядку, а для задання поверхні були вибрані поверхні другого роду Бізьє;
3. Обрано засоби для розробки програмного забезпечення, а саме платформа .Net, об’єктно-орієнтована мова програмування C#, середовище програмування Visual Studio 2019 Community та технологія WPF з декларативною мовою розмітки XAML;
4. Створено структуру програмного забезпечення, що використовує модель представлення MVP, бізнес логіка якої реалізована як окрема бібліотека для можливого подальшого використання та підміни інтерфейсу користувача;
5. Розроблено алгоритмічну базу для реалізації деформації зображення вибраними засобами та методами, створення морфінгу двох зображень та нанесення їх на поверхню;
6. Розроблено програмний продукт для створення анімації морфінгу та перенесення її на задану поверхню.

Програмне забезпечення розроблено мовою програмування C# для платформи .Net в середовищі Visual Studio 2019.

В ході дипломної роботи проведено дослідження деформації растрових зображень для нанесення їх на поверхні, дослідження та порівняння існуючих методів морфінгу зображення, а також розроблено програмний продукт, основними функціями якого є генерація послідовності зображення вибраним користувачем методом морфінгу, відображення анімації морфінгу, побудови заданої поверхні та нанесення на неї анімації морфінгу двох зображень.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Microsoft Corporation Paint 3D [Електронний ресурс] – https://www.microsoft.com/uk-ua/p/paint-3d/9nblggh5fv99?activetab=pivot:overviewtab.
2. Adobe Photoshop CC [Електронний ресурс] – https://www.adobe.com/ua/products/photoshop.html.
3. 3DS MAX. 3D modeling and rendering software for design visualization, games, and animation [Електронний ресурс] – Режим доступу: https://www.autodesk.ru/products/3ds-max/overview.
4. Squilz Morph for Windows [Електронний ресурс] – Режим доступу: https://sqirlz-morph.en.softonic.com/#tab-review.
5. Magdil Delport Morphing in Two Dimensions: Image Morphing. Thesis presented in partial fulfilment of the requirements for the degree of Master of Science at Stellenbosch University / Magdil Delport // Western Cape Stellenbosch University, 2007. – 99 с.
6. Lee S-Y, Wolberg G., Shin S. Polymorph: Morphing Among Multiple Images. IEEE Computer Graphics and Applications / Lee S-Y, Wolberg G., Shin S. // New York: National Computer Graphics Association, 1998. – 15 с.
7. Аушева Н. М. Лінійні перетворення. Методичні вказівки до виконання лабораторних робіт з курсу “Комп’ютерна графіка”. / Аушева Наталія Миколаївна // К: ІВЦ Видавництво «Політехніка», 2005. – 24 с.
8. Заславский А. А. Геометрические преобразования / Заславский Алексей Александрович // М: МЦНМО, 2004. – 86 с.
9. Gomes J, Costa B, Darsa L, Velho L Warping and Morphing of Graphical Objects / Gomes J, Costa B, Darsa L, Velho L // Morgan Kaufmann Publisher, Inc, San Fransisco, 1999. – 168 с.
10. M´alkov´a M. Morphing of geometrical objects in boundary representation / Martina M´alkov´a // Pilsen: University of West Bohemia, 2010. – 63 с.

11. Joseph Choma Morphing: A Guide to Mathematical Transformations for Architects and Designers / Joseph Choma // Laurence King Publishing, 2015. – 232 с.
12. Рик Пэрент Компьютерная анимация. Теория и алгоритмы. / Рик Пэрент // М.: Кудиц-Образ, 2004. – 554 с.
13. Д. Роджерс Алгоритмические основы машинной графики / Д. Роджерс // М.: Мир, 2001. – 512 с.
14. Д. Роджерс, Дж. Адамс Математические основы машинной графики / Д. Роджерс, Дж. Адамс // М.: Мир, 1998. – 601 с.
15. Дёмин А. Ю., Кудинов А. В. Компьютерная графика / Дёмин А. Ю., Кудинов А. В. // [Електронний ресурс] – Режим доступу: http://compgraph.tpu.ru/index.html.
16. А. В. Скворцов Триангуляция Делоне и её применение / А. В. Скворцов // Томск: Изд-во Том, 2002. – 128 с.
17. Oana-Ancuta Buzura Interpolation Algorithm using Catmull-Rom Spline Function and Operator / Oana-Ancuta Buzura, Rodica Holonec, Petru Marius // Romania: Mediamira Science Publisher, 2016. – 10 с.
18. Шикин Е.В., Плис Л.И. Кривые и поверхности на экране компьютера. Руководство по сплайнам для пользователей. / Шикин Е.В., Плис Л.И // Москва, 1996. – С. 140-148.
19. Рихтер Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 4-е изд. / Рихтер Дж. // СПб.: Питер, 2013. – 896 с.
20. Matthew MacDonald Pro WPF 4.5 in C#. Windows Presentation Foundation in .Net 4.5 Fourth Edition. / Matthew MacDonald // Apress, 2012. – 1112 с.

ДОДАТОК А

Деформація растрових зображень для нанесення на поверхні

Специфікація

УКР.НТУУ”КПІ імені Ігоря Сікорського”_ТЕФ_АПЕПС_ТР5157_19Б

Аркушів 2

Київ 2019

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ ”_ТЕФ_АПЕПС_ТР5157_19Б	Записка.docx	Текстова частина дипломної роботи
Компоненти		
УКР.НТУУ”КПІ ”_ТЕФ_АПЕПС_ТР5157_19Б 12-1	GeneralModelLibrary.dll	Бібліотека для генерації деформованих зображень
УКР.НТУУ”КПІ ”_ТЕФ_АПЕПС_ТР5157_19Б 12-2	ImageWarpingApp.exe	Виконуваний файл інтерфейсу користувача

ДОДАТОК Б

Деформація растрових зображень для нанесення на поверхні

Текст програми

УКР.НТУУ”КПІ імені Ігоря Сікорського”_ТЕФ_АПЕПС_ТР5157_19Б
12-1

Аркушів 7

Київ 2019

```
using GeneralModelLibrary.Morphing.Extensions;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.IO;
using System.Linq;

namespace GeneralModelLibrary.Morphing {
    public class MorphingModel : IMorphingOperations, ISurfaceCreator {
        public ControlPoints ControlPoints { get; set; }
        public List<Polygon> ChildPolygons { get; private set; } = new List<Polygon>();
        public List<Polygon> LeopardPolygons { get; private set; } = new List<Polygon>();
        public List<Triangle> ChildTriangles { get; private set; } = new List<Triangle>();
        public List<Triangle> LeopardTriangles { get; private set; } = new List<Triangle>();

        private readonly string generatedFilesPath;
        private readonly string sourceImagesPath;
        private readonly string surfaceFolderPath;
        private readonly string catmullRomFolder;
        private readonly string triangulationFolder;
        private readonly string triangulationMethodName;

        public MorphingModel(string generatedFilesPath, string sourceImagesPath, string surfaceFolderPath,
            string catmullRomFolder, string triangulationFolder, string triangulationMethodName) {
            this.generatedFilesPath = generatedFilesPath;
            this.sourceImagesPath = sourceImagesPath;
            this.surfaceFolderPath = surfaceFolderPath;
            this.catmullRomFolder = catmullRomFolder;
            this.triangulationFolder = triangulationFolder;
            this.triangulationMethodName = triangulationMethodName;

            if(!Directory.Exists(generatedFilesPath)) {
                Directory.CreateDirectory(generatedFilesPath);
            }

            string catmulRomPath = generatedFilesPath + catmullRomFolder;
            if(!Directory.Exists(catmulRomPath)) {
                Directory.CreateDirectory(catmulRomPath);
            }
            string triangulationPath = generatedFilesPath + triangulationFolder;
            if(!Directory.Exists(triangulationPath)) {
                Directory.CreateDirectory(triangulationPath);
            }
            string surfaceCatmulRom = surfaceFolderPath + catmullRomFolder;
            string surfaceBezier = surfaceFolderPath + triangulationFolder;
            if(!Directory.Exists(surfaceFolderPath)) {
                Directory.CreateDirectory(surfaceFolderPath);
            }

            if(!Directory.Exists(surfaceCatmulRom)) {
                Directory.CreateDirectory(surfaceCatmulRom);
            }

            if(!Directory.Exists(surfaceBezier)) {
                Directory.CreateDirectory(surfaceBezier);
            }

            LoadControlPoints();
        }

        public void SetControlPointsForChild(ICollection<ControlPoint> listOfPoints) {
            ControlPoints.ChildControlPoints.Clear();
            ControlPoints.ChildControlPoints.AddRange(listOfPoints);
        }

        public void SetControlPointsForLeopard(ICollection<ControlPoint> listOfPoints) {
            ControlPoints.LeopardControlPoints.Clear();
            ControlPoints.LeopardControlPoints.AddRange(listOfPoints);
        }

        public void Save() {
            ControlPoints.Serialize();
        }

        private void LoadControlPoints() {
            ControlPoints = ControlPoints.DeserializeControlPoints();
            ChildPolygons = SetDefaultPolygons(ControlPoints.ChildControlPoints);
```

```
            ChildTriangles = SetDefaultTriangleForTriangulation(ChildPolygons);
            LeopardPolygons = SetDefaultPolygons(ControlPoints.LeopardControlPoints);
            LeopardTriangles = SetDefaultTriangleForTriangulation(LeopardPolygons);
        }

        public void CreateGridForCellByCatmullRom(Bitmap picture, Bitmap newPicture, List<Polygon> startPolygons,
List<Polygon> finalPolygons, int n = 5) {
            Polygon startCurrentPolygon = startPolygons[n];
            Polygon leftStartPolygon = startPolygons
                .Where(p => p.P3 == startCurrentPolygon.P2 && p.P4 == startCurrentPolygon.P1).FirstOrDefault() ??
startCurrentPolygon;
            Polygon rightStartPolygon = startPolygons
                .Where(p => p.P2 == startCurrentPolygon.P3 && p.P1 == startCurrentPolygon.P4).FirstOrDefault() ??
startCurrentPolygon;

            Polygon endCurrentPolygon = finalPolygons[n];
            Polygon leftEndPolygon = finalPolygons
                .Where(p => p.P3 == endCurrentPolygon.P2 && p.P4 == endCurrentPolygon.P1)
                .FirstOrDefault() ?? endCurrentPolygon;
            Polygon rightEndPolygon = finalPolygons
                .Where(p => p.P2 == endCurrentPolygon.P3 && p.P1 == endCurrentPolygon.P4)
                .FirstOrDefault() ?? endCurrentPolygon;

            double maxSide = GeneralGraphicsOperations.GetMaxLateralSide(startCurrentPolygon, leftStartPolygon,
rightStartPolygon);

            double numberOfRows = Math.Ceiling(maxSide * 2);
            double step = 1 / numberOfRows;
            Point[,] rowsForStartLateralSides = new Point[4, (int)numberOfRows];
            for(int i = 0; i < numberOfRows; i++) {
                rowsForStartLateralSides[0, i] = GeneralGraphicsOperations.LinearInterpolation(leftStartPolygon.P1,
leftStartPolygon.P2, i * step);
                rowsForStartLateralSides[1, i] =
GeneralGraphicsOperations.LinearInterpolation(startCurrentPolygon.P1, startCurrentPolygon.P2, i * step);
                rowsForStartLateralSides[2, i] =
GeneralGraphicsOperations.LinearInterpolation(startCurrentPolygon.P4, startCurrentPolygon.P3, i * step);
                rowsForStartLateralSides[3, i] = GeneralGraphicsOperations.LinearInterpolation(rightStartPolygon.P4,
rightStartPolygon.P3, i * step);
            }

            Point[,] rowsForEndLateralSides = new Point[4, (int)numberOfRows];
            for(int i = 0; i < numberOfRows; i++) {
                rowsForEndLateralSides[0, i] = GeneralGraphicsOperations.LinearInterpolation(leftEndPolygon.P1,
leftEndPolygon.P2, i * step);
                rowsForEndLateralSides[1, i] = GeneralGraphicsOperations.LinearInterpolation(endCurrentPolygon.P1,
endCurrentPolygon.P2, i * step);
                rowsForEndLateralSides[2, i] = GeneralGraphicsOperations.LinearInterpolation(endCurrentPolygon.P4,
endCurrentPolygon.P3, i * step);
                rowsForEndLateralSides[3, i] = GeneralGraphicsOperations.LinearInterpolation(rightEndPolygon.P4,
rightEndPolygon.P3, i * step);
            }

            double lengthOfTopSide = GeneralGraphicsOperations.GetLength(startCurrentPolygon.P1,
startCurrentPolygon.P4);
            double lengthOfBottomSide = GeneralGraphicsOperations.GetLength(startCurrentPolygon.P2,
startCurrentPolygon.P3);
            var numberOfColumns = Math.Ceiling(Math.Max(lengthOfTopSide, lengthOfBottomSide) * 2.5);

            Color[,] cellPicture = new Color[(int)numberOfRows, (int)numberOfColumns];
            Point[,] matrixOfPointsForStartPolygon = new Point[(int)numberOfRows, (int)numberOfColumns];
            for(int i = 0; i < numberOfRows; i++) {
                for(int j = 0; j < numberOfColumns; j++) {
                    matrixOfPointsForStartPolygon[i, j] = new Point {
                        X = (int)Math.Floor(GeneralGraphicsOperations.CatmullRomFunction(rowsForStartLateralSides[0,
i].X, rowsForStartLateralSides[1, i].X, rowsForStartLateralSides[2, i].X, rowsForStartLateralSides[3, i].X, j /
numberOfColumns)),
                        Y = (int)Math.Floor(GeneralGraphicsOperations.CatmullRomFunction(rowsForStartLateralSides[0,
i].Y, rowsForStartLateralSides[1, i].Y, rowsForStartLateralSides[2, i].Y, rowsForStartLateralSides[3, i].Y, j /
numberOfColumns)),
                    };
                }
            }

            Point[,] matrixOfPointsForEndPolygon = new Point[(int)numberOfRows, (int)numberOfColumns];
            for(int i = 0; i < numberOfRows; i++) {
                for(int j = 0; j < numberOfColumns; j++) {
                    matrixOfPointsForEndPolygon[i, j] = new Point {
```

```
                    X = (int)Math.Floor(GeneralGraphicsOperations.CatmullRomFunction(rowsForEndLateralSides[0,
i].X, rowsForEndLateralSides[1, i].X, rowsForEndLateralSides[2, i].X, rowsForEndLateralSides[3, i].X, j /
numberOfColumns)),
                    Y = (int)Math.Floor(GeneralGraphicsOperations.CatmullRomFunction(rowsForEndLateralSides[0,
i].Y, rowsForEndLateralSides[1, i].Y, rowsForEndLateralSides[2, i].Y, rowsForEndLateralSides[3, i].Y, j /
numberOfColumns)),
                };
            }
        }
        for(int i = 0; i < numberOfRows; i++) {
            for(int j = 0; j < numberOfColumns; j++) {
                int x = Math.Abs(matrixOfPointsForStartPolygon[i, j].X).Clamp(0, newPicture.Width - 1);
                int y = Math.Abs(matrixOfPointsForStartPolygon[i, j].Y).Clamp(0, newPicture.Height - 1);
                int x1 = Math.Abs(matrixOfPointsForEndPolygon[i, j].X).Clamp(0, picture.Width - 1);
                int y1 = Math.Abs(matrixOfPointsForEndPolygon[i, j].Y).Clamp(0, picture.Height - 1);
                newPicture.SetPixel(x1, y1, picture.GetPixel(x, y));
            }
        }
    }

    private void CreatePictureForGridBySecondBezier(Bitmap startPicture, Bitmap targetPicture, Triangle
sourceTriangle, Triangle targetTriangle) {
        SetPixelsForTrianleFromStartPictureToTarget(startPicture, targetPicture,
           sourceTriangle.P1, sourceTriangle.P3, sourceTriangle.P2,
           targetTriangle.P1, targetTriangle.P3, targetTriangle.P2);
        SetPixelsForTrianleFromStartPictureToTarget(startPicture, targetPicture,
           sourceTriangle.P2, sourceTriangle.P1, sourceTriangle.P3,
           targetTriangle.P2, targetTriangle.P1, targetTriangle.P3);
        SetPixelsForTrianleFromStartPictureToTarget(startPicture, targetPicture,
           sourceTriangle.P1, sourceTriangle.P2, sourceTriangle.P3,
           targetTriangle.P1, targetTriangle.P2, targetTriangle.P3);
    }
    private void SetPixelsForTrianleFromStartPictureToTarget(Bitmap startPicture, Bitmap targetPicture,
        Point sourceP1, Point sourceP2, Point sourceP3, Point targetP1, Point targetP2, Point targetP3) {

        Point sourceMiddlePoint = GeneralGraphicsOperations.GetCenterPointBetweenTwo(sourceP1, sourceP3);
        Point targetMiddlePoint = GeneralGraphicsOperations.GetCenterPointBetweenTwo(targetP1, targetP3);
        double maxLength = Math.Ceiling(
            Math.Max(
                GeneralGraphicsOperations.GetLength(sourceP2, sourceMiddlePoint),
                GeneralGraphicsOperations.GetLength(targetP2, targetMiddlePoint)
                )
            );

        Point sourcePoint, targetPoint;
        int numberOfCurve = (int)maxLength;
        double step = 1 / maxLength;
        for(int i = 0; i < numberOfCurve; i++) {
            for(int j = 0; j < numberOfCurve; j++) {
                sourcePoint = GeneralGraphicsOperations.
                    FractionalFormCurve(sourceP1, 1,
                        GeneralGraphicsOperations.LinearInterpolation(sourceMiddlePoint, sourceP2, i * step), 2,
                        sourceP3, 1, j * step
                    );
                targetPoint = GeneralGraphicsOperations.
                    FractionalFormCurve(targetP1, 1,
                        GeneralGraphicsOperations.LinearInterpolation(targetMiddlePoint, targetP2, i * step), 2,
                        targetP3, 1, j * step
                    );
                int x = sourcePoint.X.Clamp(0, startPicture.Width - 1);
                int y = sourcePoint.Y.Clamp(0, startPicture.Height - 1);
                int x1 = targetPoint.X.Clamp(0, targetPicture.Width - 1);
                int y1 = targetPoint.Y.Clamp(0, targetPicture.Height - 1);
                if(targetPicture.GetPixel(x1, y1) == Color.FromArgb(0, 0, 0, 0)) {
                    targetPicture.SetPixel(x1, y1, startPicture.GetPixel(x, y));
                }
            }
        }
    }

    public void PolygonsMorphing() {
        Bitmap ChildPicture = new Bitmap(sourceImagesPath + @"\child.png");
        var sequenceOfGrids = GenerateSequenceOfGrids(10, ControlPoints.ChildControlPoints,
ControlPoints.LeopardControlPoints,
            (controlPoints) => SetDefaultPolygons(controlPoints));
        for(int i = 0; i < sequenceOfGrids.Count; i++) {
            Bitmap newChildPicture = new Bitmap(ChildPicture.Width, ChildPicture.Height);
```

```
                for(int j = 0; j < 28; j++) {
                    CreateGridForCellByCatmullRom(ChildPicture, newChildPicture, ChildPolygons, sequenceOfGrids[i],
j);
                }
                newChildPicture.Save(generatedFilesPath + catmullRomFolder + $@"\child_{i}.png");
            }

            Bitmap LeopardPicture = new Bitmap(sourceImagesPath + @"\leopard_Resized.png");
            sequenceOfGrids = GenerateSequenceOfGrids(10, ControlPoints.LeopardControlPoints,
ControlPoints.ChildControlPoints,
                (controlPoints) => SetDefaultPolygons(controlPoints));
            for(int i = 0; i < sequenceOfGrids.Count; i++) {
                Bitmap newLeopardPicture = new Bitmap(LeopardPicture.Width, LeopardPicture.Height);
                for(int j = 0; j < 28; j++) {
                    CreateGridForCellByCatmullRom(LeopardPicture, newLeopardPicture, LeopardPolygons,
sequenceOfGrids[i], j);
                }
                newLeopardPicture.Save(generatedFilesPath + catmullRomFolder + $@"\leopard_{i}.png");
            }
        }

        public void TriangulationMorphing() {
            Bitmap ChildPicture = new Bitmap(sourceImagesPath + @"\child.png");
            List<List<Triangle>> sequenceOfGrids = GenerateSequenceOfGrids(10, ControlPoints.ChildControlPoints,
ControlPoints.LeopardControlPoints,
                controlPoints => SetDefaultPolygons(controlPoints))
                    .Select(polygonGrid => SetDefaultTriangleForTriangulation(polygonGrid)).ToList();
            for(int i = 0; i < sequenceOfGrids.Count; i++) {
                Bitmap newChildPicture = new Bitmap(ChildPicture.Width, ChildPicture.Height);
                for(int j = 0; j < ChildTriangles.Count; j++) {
                    CreatePictureForGridBySecondBezier(ChildPicture, newChildPicture, ChildTriangles[j],
sequenceOfGrids[i][j]);
                }
                newChildPicture.Save(generatedFilesPath + triangulationFolder + $@"\child_{i}.png");
            }

            Bitmap LeopardPicture = new Bitmap(sourceImagesPath + @"\leopard_Resized.png");
            sequenceOfGrids = GenerateSequenceOfGrids(10, ControlPoints.LeopardControlPoints,
ControlPoints.ChildControlPoints,
                controlPoints => SetDefaultPolygons(controlPoints))
                    .Select(polygonGrid => SetDefaultTriangleForTriangulation(polygonGrid)).ToList();
            for(int i = 0; i < sequenceOfGrids.Count; i++) {
                Bitmap newLeopardPicture = new Bitmap(LeopardPicture.Width, LeopardPicture.Height);
                for(int j = 0; j < ChildTriangles.Count; j++) {
                    CreatePictureForGridBySecondBezier(LeopardPicture, newLeopardPicture, LeopardTriangles[j],
sequenceOfGrids[i][j]);
                }
                newLeopardPicture.Save(generatedFilesPath + triangulationFolder + $@"\leopard_{i}.png");
            }
        }
        public void CreateSurfaceMorphing(string methodName, PointsForBezierSurface pointsForBezierSurface,
            int numberOfImages, int newWidth, int newHeight, double projectionDistance) {
            List<Bitmap> childImages = new List<Bitmap>();
            List<Bitmap> leopardImages = new List<Bitmap>();

            string path;
            childImages.Add(new Bitmap(sourceImagesPath + @"\child.png"));
            if(methodName == triangulationMethodName) {
                path = triangulationFolder;
            } else {
                path = catmullRomFolder;
            }

            for(int i = 0; i < numberOfImages; i++) {
                childImages.Add(new Bitmap(generatedFilesPath + path + $@"\child_{i}.png"));
            }

            for(int i = numberOfImages - 1; i >= 0; i--) {
                leopardImages.Add(new Bitmap(generatedFilesPath + path + $@"\leopard_{i}.png"));
            }
            leopardImages.Add(new Bitmap(sourceImagesPath + @"\leopard_Resized.png"));

            List<int> rotationAngles = new List<int>();
            for(int i = -15; i <= 15; i += 5) {
                rotationAngles.Add(i);
            }
```

```
            foreach(var angle in rotationAngles) {
                List<Bitmap> leopardSurfaceImages = new List<Bitmap>();
                List<Bitmap> childSurfaceImages = new List<Bitmap>();

                Point3D[,] surfaceMatrix = GetSurfaceWithSpecifiedSize(childImages[0].Width, childImages[0].Height,
pointsForBezierSurface);

                for(int index = 0; index < childImages.Count; index++) {
                    Bitmap currentImage = childImages[index];
                    SetSurfacePictures(surfaceMatrix, currentImage, childSurfaceImages, newWidth, newHeight,
projectionDistance, angle);
                    currentImage = leopardImages[index];
                    SetSurfacePictures(surfaceMatrix, currentImage, leopardSurfaceImages, newWidth, newHeight,
projectionDistance, angle);
                }

                for(int i = 0; i < childSurfaceImages.Count; i++) {
                    childSurfaceImages[i].Save(surfaceFolderPath + path + $@"\childSurface_{angle}_{i}.png");
                    leopardSurfaceImages[i].Save(surfaceFolderPath + path + $@"\leopardSurface_{angle}_{i}.png");
                }
            }
        }

        public Point3D[,] GetSurfaceWithSpecifiedSize(int width, int height, PointsForBezierSurface
pointsForBezierSurface) {
            return GeneralGraphicsOperations.GetSurfaceWithSpecifiedSize(width, height, pointsForBezierSurface);
        }
        public void SetSurfacePictures(Point3D[,] surface, Bitmap currentImage, List<Bitmap> surfaceImages, int
newWidth, int newHeight, double projectionDistance, double rotationAngle) {
            var surfaceMatrix = surface.RotateSurfaceAroundOy(rotationAngle);
            Bitmap finalImage = new Bitmap(newWidth, newHeight);

            for(int i = 0; i < currentImage.Width; i++) {
                for(int j = 0; j < currentImage.Height; j++) {
                    var point2D = GeneralGraphicsOperations.GetProjectionOf3DPointInto2D(surfaceMatrix[i, j],
projectionDistance);
                    point2D.X += finalImage.Width / 2;
                    point2D.Y += finalImage.Height / 2;

                    if(!point2D.IsOutsideOfImage(finalImage)) {
                        if(finalImage.GetPixel(point2D.X, point2D.Y) == Color.FromArgb(0, 0, 0, 0)) {
                            finalImage.SetPixel(
                                point2D.X,
                                point2D.Y,
                                currentImage.GetPixel(i, j)
                            );
                        }
                    }
                }
            }
            surfaceImages.Add(finalImage);
        }


        public void Morphing(string name) {
            if(name == triangulationMethodName) {
                TriangulationMorphing();
            } else {
                PolygonsMorphing();
            }
        }

        public List<List<T>> GenerateSequenceOfGrids<T>(int numberOfGrids, List<ControlPoint> sourceControlPoints,
List<ControlPoint> targetControlPoints, Func<List<ControlPoint>, List<T>> setGrid) {
            List<List<T>> result = new List<List<T>>();
            List<ControlPoint> sourceControlPointsCopy = new List<ControlPoint>(sourceControlPoints);
            for(int i = 0; i < numberOfGrids; i++) {
                double step = i / (numberOfGrids - 1.0);
                sourceControlPointsCopy = sourceControlPointsCopy
                    .Select(point => new ControlPoint {
                        Point = GeneralGraphicsOperations.LinearInterpolation(point.Point,
targetControlPoints[point.Index].Point, step),
                        Index = point.Index
                    }).ToList();
                List<T> grid = setGrid(sourceControlPointsCopy);
                result.Add(grid);
            }
```

```
            return result;
        }

        public List<Triangle> SetDefaultTriangleForTriangulation(List<Polygon> polygons) {
            var result = new List<Triangle>();

            foreach(var polygon in polygons) {
                result.Add(new Triangle {
                    P1 = polygon.P1,
                    P2 = polygon.P2,
                    P3 = polygon.P3
                });
                result.Add(new Triangle {
                    P1 = polygon.P3,
                    P2 = polygon.P4,
                    P3 = polygon.P1
                });
            }

            return result;
        }
    }
}
```

ДОДАТОК В

Деформація растрових зображень для нанесення на поверхні

Опис програми

УКР.НТУУ”КПІ імені Ігоря Сікорського”_ТЕФ_АПЕПС_ТР5157_19Б
13-1

Аркушів 9

Київ 2019

АНОТАЦІЯ

Розроблена система містить дві компоненти. Перша з них – це бібліотека класів, що реалізує весь функціонал для деформації зображення, генерації послідовності зображень та містить реалізацію методів для розрахунку простих геометричних операцій. Бібліотека носить назву GeneralModelLibrary.dll, розроблена для платформи .Net і може бути використана в подальшому як звичайна бібліотека DLL.

Другим компонентом є виконуваний файл ImageWarpingApp.exe, що представляє реалізацію інтерфейсу користувача взаємодії з розроблено бібліотекою для вирішення завдань генерації послідовності деформованих зображень, створення анімації морфінгу та накладання її на задану поверхню. ImageWarpingApp.exe розроблений мовою C# для платформи .Net із застосуванням MVP патерну і використовує технологію WPF та декларативну мову розмітки XAML для створення зовнішнього вигляду інтерфейсу.

Програмне забезпечення було розроблене за допомогою програмного середовища Visual Studio Community 2019 об’єктно-орієнтованою мовою C#. Для створення графічного інтерфейсу було використано технологіє WPF.

ЗМІСТ

1. Загальні відомості..... 4
2. Функціональне призначення 5
3. Опис логічної структури..... 6
4. Технічні засоби, що використовуються 7
5. Виклик і завантаження..... 8
6. Вхідні і вихідні дані 9

ЗАГАЛЬНІ ВІДОМОСТІ

Розроблений програмний продукт працює на базі операційної системи Windows 10. Для функціонування не потребує ніяких попередньо встановлених програмних засобів.

Програмний продукт розроблений на об’єктно-орієнтованій мові програмування C# за допомогою середовища розробки Visual Studio Community 2019. Також було використано технологію WPF та мову розмітки XAML для створення графічного інтерфейсу користувача.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Програмна система вирішує завдання деформації зображення, морфінгу двох зображень на площині та нанесення анімації морфінгу на поверхню.

Програмна система призначена для деформації растрових зображень, генерації послідовності трансформованих зображень, анімації морфінгу, задання поверхні та нанесення анімації на поверхні.

Функціональних обмежень на використання розроблених компонентів не має.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Перед початком генерації деформованих растрових зображень, користувачеві необхідно у спеціальному вікні задати контрольні точки для зображень, таким чином виділивши однотипні частини зображення, як наприклад очі, ніс, рот тощо.

Після цього, програма, отримавши контрольні точки, може деформувати зображення для відповідних сіток та створити анімацію морфінгу. Для накладання цієї анімації необхідно задати поверхню, після чого програма матиме можливість деформувати зображення для нанесення їх на поверхню.

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Розроблений програмний продукт працює на базі платформи .Net та операційної системи Windows 10. Система не потребує додаткових засобів для функціонування.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Розроблений програмний додаток не потрібно встановлювати на персональний комп’ютер. Для роботи з ним необхідно просто завантажити заархівований файл, розархівувати його та запустити виконуваний файл з розширенням .exe.

Система не потребує додаткових сторонніх програмний засобів, займає невеликий обсяг пам’яті та має невеликі системні вимоги. Для видалення програми з комп’ютера достатньо видалити папку, що містить виконуваний та інші файли.

ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними для програмної системи є набір контрольних точки для обох зображень та задані користувачем вершини характеристичних трикутників поверхні.

Вихідними даними є набір деформованих зображень, анімація морфінгу на площині та на заданій поверхні.

ДОДАТОК Г

Деформація растрових зображень для нанесення на поверхні

Апробація

УКР.НТУУ”КПІ імені Ігоря Сікорського”_ТЕФ_АПЕПС_ТР5157_19Б

Аркушів 4

Київ 2019

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

СУЧАСНІ ПРОБЛЕМИ НАУКОВОГО ЗАБЕЗПЕЧЕННЯ ЕНЕРГЕТИКИ

Матеріали XVII Міжнародної науково-практичної конференції молодих вчених та студентів м. Київ, 23-26 квітня 2019 року,

ТОМ 2

Київ- 2019

WEB система моніторингу поширення забруднення водної акваторії внаслідок аварій. 77
ПАВЛІЧЕНКО В.О., студент гр. ТМ-51
Керівник - проф., д.т.н. Бадаєв Ю.І.
Інформаційне середовище кафедри на базі Offcie 365. 78
КУПРІЯНОВ І.С., студент гр. ТР-51
Керівник - доц., к.т.н. Крамар Ю.М.
Геоінформаційна WEB система як інструмент проведення аналізу та візуалізації стану водних екосистем. 79
КРИВОКОНЬ Є.О., студент гр. ТМ-51
Керівник - ст.викл. Гурін А.Л.
Створення електронного мультимедійного підручника з курсу "Комп'ютерна графіка". 80
КАЛІКА І.М., студент гр. ТР-51
Керівник - проф., д.т.н. Аушева Н.М.
Морфінг зображень на основі геометричних сіток. 81
ЗАКОВОРОТНИЙ О.І., студент гр. ТР-51
Керівник - проф., д.т.н. Аушева Н.М.
Створення динамічного середовища для мобільних телефонів. 82
ЗАГРЕБЕЛЬНИЙ Є.О., студент гр. ТР-51
Керівник - проф., д.т.н. Аушева Н.М.
Моделювання ізотропних поверхонь з квазіконформною заміною параметра. 83
ДЕМЧУК Д.В., студент гр. ТР-51
Керівник - ст.викл. Гурін А.Л.
Нейронні мережі для синтезу мовлення. 84
ВИШНЯК О.М., студент гр. ТР-52
Керівник - доц., к.т.н. Стативка Ю.І.
Система розпізнавання ключових слів у потоці мовлення. 85
БОРОЗЕНЕЦЬ М.Р., студент гр. ТР-52
Керівник - доц., к.т.н. Стативка Ю.І.
Хмарний сервіс швидкого прототипування. 86
БОЙКО І.В., студент гр. ТР-52
Керівник - доц., к.т.н. Демчишин А.А.
Варіанти інтерполяційної функції Гауса. 87
ГОРОДЕЦЬКИЙ М.В., студент гр. ТР-62
Керівник - доц., к.т.н. Сидоренко Ю.В.

СЕКЦІЯ №8 **ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ІНФОРМАЦІЙНИХ СИСТЕМ ТА МЕРЕЖНИХ КОМПЛЕКСІВ** 88

Практична реалізація автоматизованого налаштування динамічних реєстрів електронних інформаційних ресурсів. 89
ЧАЙКА А.Ю., мол. вчений
Керівник - доц., к.ф.-м.н. Карпенко С.Г.
Поняття реєстру інформаційних ресурсів та світові приклади реєстрів. 90
САТИР Б.О., аспірант
Керівник - доц., к.ф.-м.н. Карпенко С.Г.
Актуальність використання реєстрів інформаційних ресурсів. 91
САТИР Б.О., аспірант
Керівник - доц., к.ф.-м.н. Карпенко С.Г.
Класифікація зашумлених діагностичних сигналів. 92

УДК 004.92

Студент 4 курсу, гр. ТР-51 Заковоротний О.І.
Проф., д.т.н. Аушева Н.М.

МОРФІНГ ЗОБРАЖЕНЬ НА ОСНОВІ ГЕОМЕТРИЧНИХ СІТОК

Деформація зображення є невід'ємною частиної в багатьох індустріальних сферах, таких як рекламна індустрія, кінематограф, розробка комп'терних ігор тощо. Головною проблемою деформації зображення є правильний вибір методу та його реалізація для отримання бажаного результату в залежності від поставленої задачію Наприклад, в комп'ютерній грі рука персонажа, за якого грає користувач, повинна рухатися зберігаючі свою відносну форму [1].

Для розв'язання подібних задач можна використовувати різні методи. Одним із таких методів трансформації зображення є морфінг. Оскільки існує декілька алгоритмів морфінгу, головною ідеєю є створення програмного забезпечення для дослідження та порівняння ефективності цих алгоритмів.

Морфінг (metamorphosis) – це техніка трансформації однієї фігури в іншу. Морфінг найчастіше розглядають як технологію комп'ютерної анімації для генерації послідовності зображень, комбінація яких створює плавний перехід від початкового зображення до цільового. У даній роботі розглянуто наступні алгоритми морфінгу: cross dissolve, mesh warping, field morphing, triangulation [2].

Найпростішим алгоритмом є «cross dissolve». Він полягає у тому, що послідовність зображень отримується шляхом накладання початкового зображення на кінцеве зі зміної прозорості першого, таким чином початкове зображення поступово зникає у кінцевому.

В алгоритмі «mesh warping» для двох зображень накладається сітка, вузлами якої є контрольні точки. Необхідно трасформувати сітку першого зображення в сітку другого. Отримавши набір сіток трансформації зображення, початкове зображення деформуємо відносно сітки кінцевого і навпаки. Набір зображень послідовного трансформування зображення в сітку іншого, необхідно симетрично накласти їх один на одного [2].

Алгоритм «Field morphing» використовує пари відрізків, для того, щоб вказати об'єкти на зображені. Ці лінії будуть відігравати роль системи координат. Пара ліній на двох зображеннях буде визначати деформації відносно їх локальної системи координат.

Для морфінгу досить часто використовують метод триангуляції Делоне. Кожне із вхідних зображень розмежовується трикутною сіткою, виділяються підзображення, які деформуються в залежності від встановлених котрольних точок. Набір котрольних точок задається користувачем для кожного вхідного зображення. Цей набір використовується як вхідні данні для алгоритму триангуляції Делоне. [2]

Для проведення порівняння та аналізу вище описаних алгоритмів морфінгу було створено програмне забезпечення мовою програмування C#. Створене програмне забезпечення для методу генерує необхідну послідовність зображень, які потім накладаються одне на одного і створюють візульний ефект трансформації однієї фігури в іншу.

Перелік посилань:

1. Joseph Choma Morphing: A Guide to Mathematical Transformations for Architects and Designers [текст] // Joseph Choma – Laurence King Publishing, 2015. – 232 с.
2. Magdil Delport Morphing in Two Dimensions: Image Morphing [текст] // Magdil Delport – Western Cape Stellenbosch University 2007. – 99 с